

본 컬럼에 대한 모든 저작권은 DevGuru에 있습니다.
컬럼을 타 사이트 등에 기재 및 링크 또는 컬럼 내용을 인용 시 반드시
출처를 밝히셔야 합니다.
컬럼 들을 CD나 기타 매체로 배포하고자 할 경우 DevGuru에 동의를
얻으셔야 합니다.

© DevGuru Corporation. All rights reserved

기타 자세한 질문 사항들은 웹 게시판이나 support@devguru.co.kr으로
문의하기 바랍니다.

Device Driver Security에 대한 고찰 III

written by khealin(jgkim@devguru.co.kr)

본 칼럼은 Windows 2000/XP에서 Device Driver에 대한 보안 문제를 기술하는 연재 기사로서 파트 3에 해당한다. 지난 기사 1, 2에 이어서 이번 기사에서는 약속한대로 File을 Create 또는 Open할 때 구체적으로

Object manager, I/O Manager, Device Driver가 각각 어떤 security checking을 담당하는지에 대해서 알아보겠다. 사실 필자가 시간이 충분하면 좀 더 깊은 부분까지 본 내용을 다루겠지만 아시다시피 이 번 3번째 기사도 작년 2003년 10월에 이어 약 8개월 만에 다시 쓰는 내용일 만큼 충분한 시간이 주어지지 않는 것이(?) 못 내 아쉬울 뿐이지만 이 또한 본 필자의 게으름을 변명할 수는 없을 것 같다.

이번 칼럼을 먼저 보기 전에 지난 연재 기사 1, 2를 먼저 숙독한 다음에 본 칼럼을 보기를 권한다.

Big Picture

자, 그럼 본론으로 들어가 보자.

주지 하다시피 Windows NT계열의 O/S는 크게 Object 마다 Access에 필요한 권한을 가지고 또한 system-wide privilege를 통해 security를 구현한다. 따라서 security를 적용할 수 있는 단위는 Object인 것이다.

다음의 "그림-1 CreateFile의 경로"를 보면 File을 생성할 때의 Control 흐름과 Security Checking에 관련된 O/S의 Component를 나타내고 있다.

File을 생성/Open할 때의 User-Mode Component의 전체적 흐름을 보면 다음의 순서를 일반적으로 따른다.

1. User-mode Application이 Win32 file name을 인자로 CreateFile을 호출한다.
2. Kernel32.dll은 이 request를 Ntdll.dll로 전달한다. 이때 Win32 File Name을 NT file name으로 convert 한다.
3. Ntdll.dll은 NT file name을 인자로 native API 함수 NtCreateFile을 호출한다. 이때 Kernel-Mode로 진입이 이루어지며 이 request는 Ntoskernel.exe 내의 I/O Manager 가 처리하게 된다.

참고로 Native API에 관한 내용은 www.devguru.co.kr의 자료실 문서에서 Attack Native API 기사를 참고 하기 바란다.

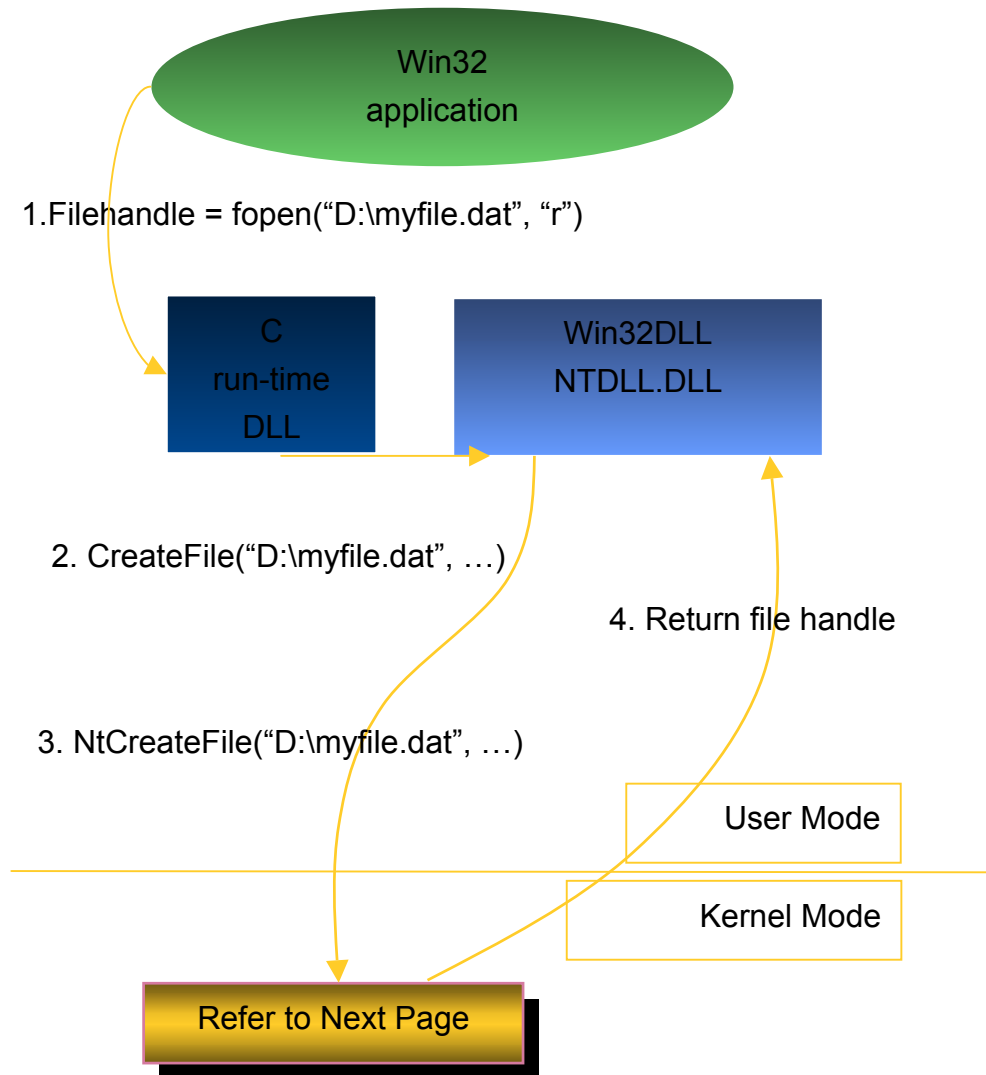


그림-1 CreateFile의 경로 1

`CreateFile` 함수 호출 시 일어나는 user-mode의 작업을 대충 살펴보았으므로 우리는 이제 kernel mode에서 어떤 일들이 일어나는지를 살펴볼 차례이다. 이해를 돕기 위해 먼저 "그림-2 CreateFile의 경로 2" 살펴보기를 바란다. 그림-2 CreateFile의 경로 2의 흐름은 대체로 다음과 같다.

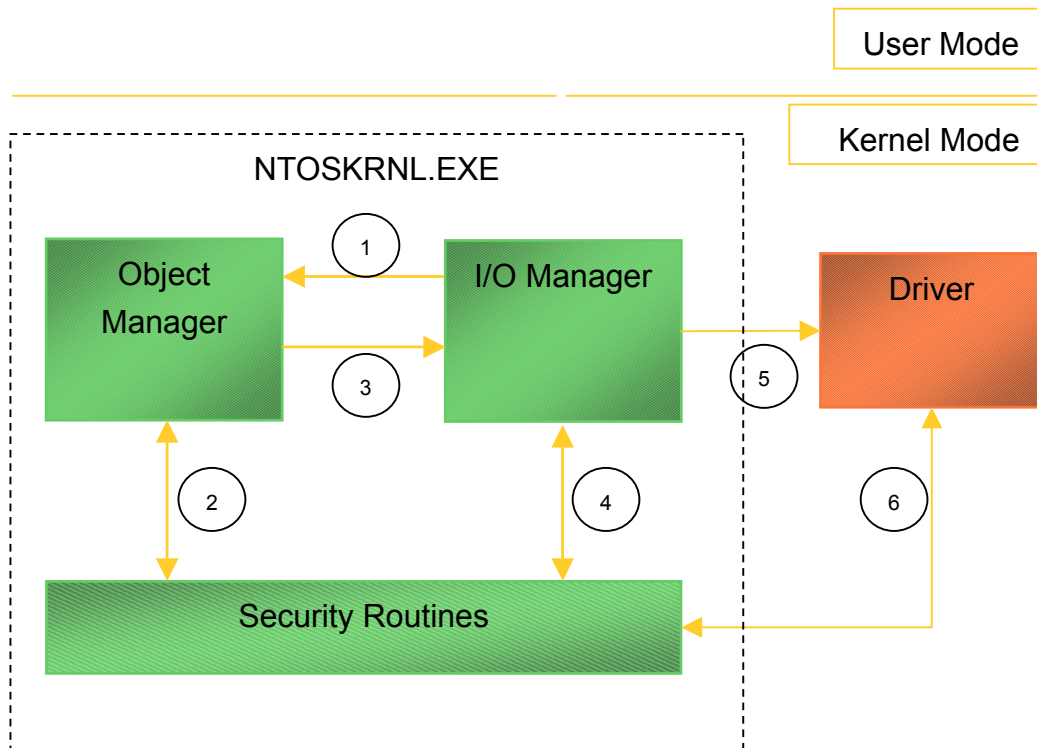


그림-2 CreateFile의 경로 2

1. The I/O Manager repackages the request into Object Manager call
2. Object Manager resolves symbolic links, checks traverse rights
3. Calls I/O Manager to process device object
4. I/O Manager checks device object security (ACLs, traversal)
5. I/O Manager creates handle, sends driver IRP_MJ_CREATE
6. Driver does additional checking as needed

위에서 열거된 6 단계의 과정은 현재 생성 또는 Open하고자 하는 대상에 따라 조금씩 상이 할 수는 있으나 대체로 위의 과정을 따른다. 이제부터 우리가 살펴볼 내용은 위에 열거된 각 과정에서 구체적으로 어떤 일들이 일어나는가이다

Under the Hood

Security Checks in the I/O Manager

Object Manager는 Event나 Mutex locks와 같은 간단한 object의 ACLs (Access Control Lists) checking을 수행한다. 그리고 Namespace를 가진 object에 대해서는 object type owner에게 Security checking을 맡긴다.

예를 들면 Device object와 file object의 type owner는 I/O Manager이므로 이들 objects에 대한 access checking은 Object Manager에서 이루어지지 않고 I/O Manager가 담당하는 것이다. 따라서 Object Manager는 I/O Manager에게 access checking을 위임한다. 그리고 다시 I/O Manager는 해당 object가 Device Name Space에 속한 object라면 Access Checking을 다시 해당 Object에 대한 Device Driver에게 위임한다.

즉 다음의 예를 보자.

Path	Description
WDeviceWDeviceName	Device object for DeviceName
WDeviceWDeviceNameW	Top-level directory on DeviceName
WDeviceWDeviceNameWFile	File on DeviceName

표-1

"WDeviceWDeviceName"를 open할 때는 I/O Manager가 type owner이므로 I/O Manager가 security checking을 수행한다.

그러나 다음의 두 path들은 특정 name space에 존재하는 object들로 간주되어 WDeviceWDeviceNameW 와 WDeviceWDeviceNameWFile를 open할 때는 type owner가 해당 driver로 인식된다. 따라서 해당 드라이버가 security checking을 수행하여야 한다.

Security Checks in the I/O Manager

이제 Security Checking에 있어서 I/O Manager의 역할에 대해서 한 번 살펴 보도록 하자. I/O Manager가 담당하는 부분은 크게 다음의 두 가지로 나누어 볼 수 있다. 물론 지금부터 논의되는 사항은 해당 type owner가 I/O Manager인 경우이다.

첫째, I/O Manager는 생성 또는 Open하고자 하는 file or Device의 handle을 만들 때 process의 access token과 object의 rights를 checking하여 User에게 허가된 권한을 handle에 저장한다. 그리고 나중에 이 handle을 사용하여 I/O 작업이 일어나면 I/O Manager는 handle에 저장된 access right을 가지고 I/O를 진행하는 process가 requested I/O를 수행 할 만큼의 권한이 있는지를 check한다.

둘째, I/O Manager는 create하고자 하는 file name 또는 device name을 parse할 때 traverse right도 check한다.

즉, 다음의 "WDeviceWFloppy0WDirectoryWFile.txt" file을 create하기 위해서는 해당 process는 각각 다음의 3가지 path들에 대한 traverse right가 있어야 한다.

WDevice

WDeviceWFloppy0

WDeviceWFloppy0WDirectory

왜냐하면 I/O Manager는 이들 path들에 대한 traverse 권한이 해당 process에게 있는지를 확인하기 때문이다. 만약에 file name이 symbolic link로 이루어져 있다면 I/O Manager는 full path name으로 변환한 다음 traverse rights를 check한다.

예를 들면 다음의 symbolic link name "WDosDeviceWA" 는 "WDeviceWFloppy0"로 변환되어 traverse rights checking이 이루어 진다. 따라서 위에서 언급된 두 가지 사항을 정리하면 다음의 그림-3과 같다고 할 수 있다.

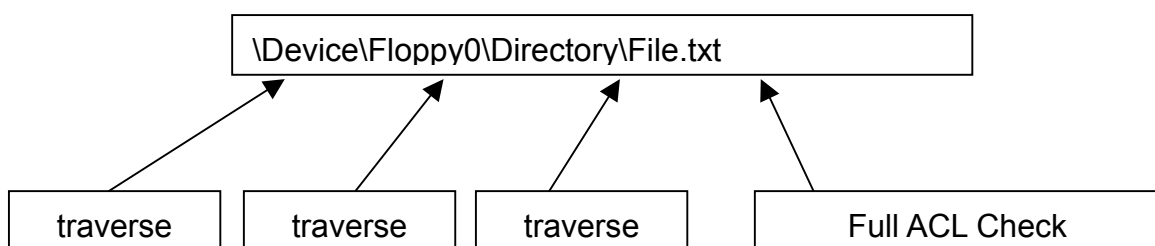


그림-3

I/O Manager는 Security Checking에 있어서 위에서 살펴본 두 가지 정도의

역할을 가진다. 이 두 가지의 경우에 대해서 좀 더 살펴볼 사항들이 존재한다.

사실 Open작업은 해당 Object에 따라 크게 두 가지로 나누어 볼 수 있다. 즉, I/O Manager가 type owner인 경우와 아닌 경우이다. 먼저 type owner인 경우를 살펴보자.

I/O Manager는 다음과 같은 "WDeviceWSerial0"(즉, COM1 port) device를 open할 때는 ACLs checking과 traverse checking을 수행한다.

(참고로 ACLs에 대한 checking은 본 기사 연재 2번을 참조 하면 된다)

일반적으로 ACLs checking의 대상 object는 named device object인 PDO가 된다. (물론 이 경우는 device stack에 현재 named device object가 PDO만 있다고 가정했을 때이다.)

따라서 unnamed object에 대한 ACLs checking은 의미가 없다.

이제 type owner가 아닌 경우인데 이때 I/O Manager의 역할은 의외로 간단하다. 즉, default로 open대한 security checking은 해당 driver에게(open 대상이 "WDeviceWDeviceNameWFile"와 같은 경우 file system driver) 위임한다. 물론 이 경우에도 traverse checking은 이루어진다.

잠시 주제의 본론에서 벗어나지만 이상의 사실들로부터 우리는 다음과 같은 사실들을 추론할 수 있다.

WDM device driver를 작성한다면 최소 2개의 device object(functional Device object와 physical device object)로 이루어진 stack을 가지게 되는데 이 때 functional device object(여러분이 만든다)의 이름을 사용하지 않는 것이 좋다. 왜냐하면 IoCreateDevice라는 DDK함수를 사용하여 device object를 만든다면 default security가 사용되어지고 이 default security는 ACLs checking이 unauthorized users에게도 pass가 되므로 unauthorized users가 여러분의 device를 마음껏 접근할 수 있기 때문이다. 그렇다면 functional device의 object가 이름이 없다면 어떻게 우리의 device는 open 될 수 있는가? 그래서 바로 Device Interface라는 새로운 device naming scheme WDM에서 나오게 된 것이다.

Driver Security Responsibility

드라이버는 자신의 name space에 대한 type owner로서 name space에 대한 security를 책임진다. 물론 이 security checking은 I/O Manager로부터

위임을 받은 것이다.

일반적으로 드라이버는 다음과 같은 항목들에 대한 device security를 책임져야 한다.

1. Creating secure device object.
2. Securing the device namespace.
3. Specifying device characteristics and security settings in INF files.
4. Defining and handling IOCTLs securely

Creating secure device object

연재 기사 2회를 참조하면 모든 device object는 device의 access를 제어하는 ACLs를 포함하고 있는 security descriptor를 가진다고 하였다. 그리고 또한 Inf file에서 SDDL를 이용하여 ACL를 설정할 수 있는 방법도 같이 설명하였다.

일반적으로 IoCreateDevice DDK 함수를 이용하여 unnamed device object를 생성하면 PnP Manager는 unnamed device object에 대해 default security descriptor를 적용한다. 따라서 이 때 반드시 우리는 해당 device에 대한 INF file에 device-specific ACL를 지정하여야 한다. 이렇게 ACL이 지정되면 PnP Manager는 지정된 ACL이 device stack에 존재하는 모든 device objects에 적용되는 것을 보장해 준다. 따라서 해당 device를 다른 processes들이 접근할 수 있게 허용하기 전에 우리의 device가 속한 device stack에 대한 securing이 이루어지는 것이다.

Securing the Device Namespace

File system driver가 아니라면 일반적으로 우리는 이 항목에 대해서 신경을 쓸 필요가 없을 것 같다. 왜냐하면 “WDeviceWSerial0”와 같은 device를 Open하는 경우 I/O Manager가 device의 type owner로서 traversal checking과 target device object의 ACL과 open을 request한 process의 access token을 checking하기 때문이다. 자세한 사항은 앞서 설명한 Security Checks in the Object Manager를 참고하기 바란다.

그리고 WDM device stack상에 device name을 가진 device object가 2개 이상 존재한다면 이들 device objects의 ALCs들은 같은 값을 가져야 한다.

왜냐하면 process는 이들 두 name중 아무 name을 사용해서 device stack을 open할 수 있기 때문이다. 따라서 특별한 아주 특별한 일이 없다면 여러분의 functional device object는 name을 가지지 않는 것이 좋다.

Device를 open할 때 한 가지 더 주의할 점을 설명한다면 exclusive device에 대한 것이다. 사실 device에 대한 exclusive 속성은 한 순간에 단 하나의 handle만을 open되게 하는 것이다. 이 속성은 IoCreateDevice 함수를 이용해서 지정할 수도 있고 또한 Inf file에 기술할 수도 있다. 이렇게 exclusive 속성이 지정되면 I/O Manager가 해당 device object의 open request가 있을 때 마다 확인하여 exclusivity를 보장하게 된다. 그러나 이러한 exclusive 속성은 device object에만 해당이 되지 device name space에 속한 files에 대해서는 해당이 되지 않는다.

마지막으로 FILE_DEVICE_SECURE_OPEN 특성에 대한 내용을 한 번 살펴보자. 이 flag는 DeviceObject->Capabilities에 저장된다. 물론, device object를 생성하거나 또한 Inf file에서 이 값을 설정할 수 있다.

일반적으로 file system driver를 제외하고는 대부분의 driver들은 device name space를 지원하지 않는다. 이러한 경우 반드시 이 flag를 설정해 주는 것이 관례이다. 만약 이 flag가 set되면 device name space에 존재하는 모든 objects에 대한 open request에 대해 device object에 대한 security descriptor를 적용한다. 그리고 이 flag에 존재 유무는 device stack의 top device object에 대해서 이루어지므로 filter driver는 반드시 이 값을 copy하여야 한다.

Specifying device characteristics and security settings in INF files

다음의 inf file snippet를 보자

```
[MyDevice.NTx86.Hw]
AddReg = MyDevice.security

[MyDevice.security]
HKR,,DeviceCharacteristics.0x10001, 0x100
HKR,,Security,,"D:P(A::GA::SY)"
```

위에서 DeviceCharacteristics entry는 두 개의 flag값을 지정하고 있는데 0x10001은 다음의 값이 DWORD type이라는 것을 나타내고 0x100은

FILE_DEVICE_SECURE_OPEN characteristic을 나타낸다.

또, Security entry는 SDDL형식으로 값이 명시되어 있는데 system-only access를 명시하고 있다.

Inf file 자체에 대한 많은 내용은 여기서 할애 할 수 없으므로 SDDL형식에 대한 내용은 여러분에게 맡기겠다.

Defining and handling IOCTLs securely

이와 관련된 내용은 분량 관계상 다음의 기회로 미루어야 할 것 같다.

Conclusion

사실 지금까지 연재된 3회의 기사 내용으로 보면 현실감이 느껴지는 부분은 별로 없으리라고 생각되어 진다. 그러나 많은 unauthorized user들은 여러분이 security에 무신경하게 작성한 driver를 이용하여 system을 crack하려고 한다는 것을 잊지 말자. 또한 여러분이 작성한 driver가 system에 치명적인 손상을 가하는 일은 없어야 할 것이다. 더 기회가 될지는 모르지만 다음 연재에서는 실제로 잘못 작성되어진 driver가 어떻게 system에 security hole을 만드느지를 살펴볼 것이다.