

Re: Accessing Ndis miniport from user mode application—one more thing

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.development.device.drivers/2007-06/msg00781.L>

- *From:* "Ron" <ron<no_spam_>@3dsp.com>
 - *Date:* Wed, 27 Jun 2007 16:50:05 -0700
-

Miki:

We went the WMI route. Yes it was a lot more coding than the IOCTL approach.
You get what you pay for.....

–Ron–

"Steve Jackowski" <SteveJackowski@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message
<news:48A19177-1A94-4964-AAF6-8F47DD9D4F14@xxxxxxxxxxxxxxxxxxxx>

Anton,

Note that I'm not recommending this approach – I don't know anything about Miki's application. He seemed to be asking for alternative approaches so I suggested one. The vast majority of application–IM interaction should use IOCTLs, so I believe we're in agreement there.

Steve

"Anton Bassov" wrote:

Steve,

The approach that you have mentioned leaves the potential security hole and is quite fragile in itself, at least if implemented the way you have described it.

If you want to make sure that only users with Admin privileges are able to send control requests to your driver (and I believe one normally has a good reason to want things work this way), your task is just straight–forward if you rely upon NdisMRegisterDevice() – all you have to control is an access to adapter's standalone device object, i.e. you have to decide who is able to call CreateFile() on it successfully. This task can be achieved by build–in

Re: Accessing Ndis miniport from user mode application—one more thing

access control mechanisms, which makes your driver compliant with the standard NT security model.

However, this is not the case with sockets – you just cannot tell the system "Please make sure that UDP port XYZ is available only to privileged users", can you? You cannot make your IM responsible for this check either – by the time a packet reaches it, context of the sending process may be already lost. Furthermore, even if you are not concerned about the security issues, what happens if UDP port that you want to use is already being used by another app????

Therefore, you have to introduce the additional TDI-level filter driver in order to make sure that no caller, apart from your app, is able to specify the given UDP port number upon its call to bind(). Please note that, in order to get into a conflict with your app, another process does not even have to explicitly specify your reserved port number upon its call to bind() – TCPIP may choose the target port accidentally if it specifies INADDR_ANY.

Therefore, your TDI filter has to foresee this possibility and be able to deal with it.

To summarize, in most cases you are about to do absolutely unnecessary additional work if you take this approach – the only type of project when it seems to be appropriate is a firewall.

Anton Bassov

"Steve Jackowski" wrote:

Hi Miki,

While I agree with Anton and the vast majority of our DNE customers use IOCTLs and we use them ourselves, there are a few (and we've done this as part of some of our products) who bypass this mechanism by using sockets. In this case, the application binds to a port and then sends/receives UDP on that port. The IM (or in our case, DNE plugin) captures UDP traffic on that port and can inject incoming to send back to the application. If the messages are small with no fragmentation, this is pretty straight forward and provides a bi-directionally indicated message rather than

Re: Accessing Ndis miniport from user mode application—one more thing

forcing the application to keep IOCTLs ready to get information from the IM. This is not for all IMs/application interaction, but there are certainly applications that can take advantage of it.

Steve

"miki" wrote:

Hi

The NDIS miniport driver my company write should support some custom features. Those features can be set and queried by user mode applications. Also the driver should be able to signal the application on certain events.

In

<http://www.ndis.com/faq/QA10290101.htm>
they describe several ways to do that:

1. Using a device control object that the miniport creates and the application IOCTL it
2. Accessing custom OIDs through IOCTL_NDIS_QUERY_GLOBAL_STATS (support only OID query – so its not good for me)
3. Accessing custom OIDS through Protocol driver
4. Accessing custom OIDS through WMI.

Can anyone tell me what is the preferred/recommended method (I tend to think that accessing the OIDs through WMI, but I would like to hear what the pros think)

I am writing both NDIS 5 and NDIS 6 miniport drivers so I prefer a solution which applies to both

Thanks
Miki

Re: Accessing Ndis miniport from user mode application—one more thing