

Re: Dynamically loading binaries in Kernel mode.

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.development.device.drivers/2006-03/msg00891.html>

- *From:* Ray Trent <rat@xxxxxxxxxxxxxx>
 - *Date:* Thu, 30 Mar 2006 14:54:20 -0800
-

If he need to write a real filesystem, I would agree wholeheartedly. However, all he needs is something that pretends to be a filesystem with 1 constant file in 1 constant directory and can serve it up. I'm not sure, but he might even be able to get away with an FS filter that just adds his file to an existing FS and pulls it up through a backdoor.

The existing samples and kits available for FS development out there make that quite a bit less daunting than it might otherwise be.

I don't know of any example PE loader sources out there, but if there is one that might push the balance in the other direction, not sure.

But remember that not only does it have to be able to do the loading, but if it doesn't properly integrate into the rest of the kernel image handling it could very easily cause problems resulting in impossible to debug bluescreens. Just to pull one example out of a hat, it better do the right thing if the hosting driver is unloaded while code is executing... say during a shutdown.

Skywing wrote:

It would be *much* easier to rewrite a PE loader than build even a very simple file system. I've done the former, and while it's not the easiest thing in the world, filesystems development pales in comparison.

"Ray Trent" <rat@xxxxxxxxxxxxxx> wrote in message
news:ObIUhrCVGHA.5332@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Ah, I see what you're trying to do now. I'll keep my theory to myself out of deference to your desire for secrecy.

Anyway, unless the place you're going to load it from is reachable via a standard filesystem and unless you load it using standard export driver mechanisms and interfaces, you're going to end up having to rewrite the entire DLL loader by yourself.

Technically this might be possible, but you wouldn't have a small task in front of you. In fact, it's probably a smaller task to develop an entire (read-only?) filesystem driver for your storage device/location/whatever and then just load an export driver the normal way by pretending it's on disk.

Re: Dynamically loading binaries in Kernel mode.

Good luck getting it to load in a particular spot in memory, though. I don't think there are any interfaces for that. But then again, maybe that's not an **actual** requirement but something you just think is a requirement. If you want an answer on this part of it, we'll probably need more information on why you think you need that.

So you see, my suggestion for how to solve your problem didn't really have anything to do with the question you actually asked.

That's why we respond to odd-looking questions with more questions.

Luis Miguel Huapaya wrote:

Well, we've considered the "separate driver" solution, but it will not do the trick for the goals we have in mind. So without too many details here is what we need to achieve:

- 1) Allocate kernel level memory (system memory) that is big enough to receive the binary image (and related static data) of the dynamic code that we want to run.
- 2) Load the code into memory. I won't get into details, but we can't have the kernel level code reading it from disk.
- 3) Do whatever address fixups, etc.. required before "executing" the code.
- 4) Execute the code.

Our dynamically linked code would publish a `GetProcAddress()` type function capable of reporting back on the entry points of every other function in the dynamic code. The `GetProcAddress()` function would undoubtedly be at a fixed offset in the binary code. We would have to figure a mechanism to report back on other functions in the binaries based on the load address.

The dynamic code must be loadable/unloadable on demand.

I can't think of any other way to achieve our intended goal. It sucks because I prefer simple solutions, but in this particular case, I don't think there is any other choice.

cheers,
Luis Miguel Huapaya

"Don Burn" wrote:

Actually, the group will answer the question without going into the details of your project. What we are looking for are the goals of you need to load binaries. There are a number of solutions including do not do that, but

Re: Dynamically loading binaries in Kernel mode.

without understanding the reasoning behind the need a lot of us find we are helping people shoot the system in the foot.

For instance in you situation, there are multiple methods you can use, kernel mode DLL's, a seperate driver with an interface handled in many ways, etc. The problem is that many people come to us with "I want to do X, how can I accomplish it", when if they stepped back one level the answer is "Use standard mechanism A".

A common example of this are people asking many different ways to have the driver communicate with the application. Once we understand that the question "how do I map a user space code section into the kernel?" is really "how do I call up to an application with an action to do?" the answer becomes obvious.

--

Don Burn (MVP, Windows DDK)
Windows 2k/XP/2k3 Filesystem and Driver
Consulting
Remove StopSpam from the email to reply

"Luis Miguel Huapaya"

<LuisMiguelHuapaya@xxxxxxxxxxxxxxxxxxxxxxxxxxxx>

wrote in message

news:52930D30-4618-4C8B-BDEF-6BE5FDE193C5@xxxxxxxxxxxxxxxxxxxx

Hi again,

I understand where you are coming from. Unfortunately there are legal reasons pertaining to ongoing patent work that would prevent me from disclosing the reasons why we (I) need to dynamically load code at the kernel level. It's unfortunately one of those situations where telling anyone about our master plan basically screws us :-)

Re: Dynamically loading binaries in Kernel mode.

So it sounds to me like I need to take this offline with a consultant, get an NDA signed and proceed with some technical questions since for the most part, the participants on this forum feel ill at ease with answering my questions without prior full disclosure (which won't happen anytime soon).

cheers,
Luis Miguel Huapaya

--
Ray

--
Ray

.