If you can't find the function, find the caller and see what the caller jumps to

devblogs.microsoft.com/oldnewthing/20130214-00

February 14, 2013

Raymond Chen

You're debugging a program and you want to set a breakpoint on some function, say, `netapi32!DsAddressToSiteNameW`, but when you execute the `bp netapi32!DsAddress-ToSiteNameW` command in the debugger, the debugger says that there is no such function.

> The *Advanced Windows Debugging* book says that the `bp` command should set a breakpoint on the function, but the debugger says that the symbol cannot be found. I used the `x netapi32!*` command to see that the debugger did find a whole bunch of symbols, and it says that the symbols were loaded (from the public symbol store), but `netapi32!Ds-AddressToSiteNameW` isn't among them. The MSDN documentation says that `DsAddress-ToSiteNameW` is in the `netapi32.dll`, but it's not there! I can't believe you guys stripped that function out of the symbol file, since it's a function that people will want to set a breakpoint on.

Okay, first let's rule out the conspiracy theory. The symbols were not stripped from the public symbols. And even if they were, that shouldn't stop you, because after all, the *loader* has to be able to find the function when it loads your program, so it's gotta be obtainable even without symbols.

Don't be helpless. You already have the tools to figure out where the function is.

Just write a program that calls the function, then load it into the debugger and see what the destination of the `call` instruction is. You don't even have to pass valid parameters to the function call, since you're never actually executing the code; you're just looking at it.

And hey looky-here, you already have a program that calls the function: The program you're trying to debug! So let's see where it goes.

```
0:001>u contoso!AwesomeFunction
…
00407352 call [contoso!__imp__DsAddressToSiteNameW (0040f104)]
…
0:001>u poi 0040f104
logoncli!DsAddressToSiteNameW:
7f014710 push ebp
7f014711 mov esp, ebp
…
```

There you go. The code for the function is in `logoncli.dll`.

What happened? How did you end up in `logoncli.dll`?

What you saw was the effect of a DLL forwarder. The code for the function `DsAddressToSiteNameW` doesn't live in `netapi32.dll`. Instead, `netapi32.dll` has an export table entry that says "If anybody comes to me asking for `DsAddressToSiteNameW`, send them to `logoncli!DsAddressToSiteNameW` instead."

Officially, the function is in `netapi32.dll` for linkage purposes, but internally the function has been forwarded to another DLL for implementation. It's like a telephone call-forwarding service for DLL functions, except that instead of forwarding telephone calls, it forwards function calls. You publish a phone number in all your marketing materials, and behind the scenes, you set up the number to forward to the phone of the person responsible for sales. That way, if that person quits, or the responsibility for selling the product changes, you can just update the call-forwarding table, and all the calls get routed to the new person.

That's what happenned here. The MSDN phone book lists the function as being in `netapi32.dll`, and whenever a call comes in, it gets forwarded to wherever the implementation happens to be. And the implementation has moved around over time, so you should continue calling `netapi32!DsAddressToSiteNameW` and let the call-forwarding do the work of getting you to the implementation.

Don't start calling `logoncli` directly, thinking that you're cutting out the middle man, or in a future version of Windows, your program may start failing with a "This number is no longer in service" error, like calling the direct office number for the previous sales representative, only to find that he left the company last month.

Raymond Chen

Follow