Restoring symbols to a stack trace originally generated without symbols

devblogs.microsoft.com/oldnewthing/20131115-00

November 15, 2013

Raymond Chen

Has this ever happened to you?

```
litware!Ordinal3+0x6042
litware!DllInstall+0x4c90
litware!DllInstall+0x4b9e
contoso!DllGetClassObject+0x93c3
contoso!DllGetClassObject+0x97a9
contoso!DllGetClassObject+0x967c
contoso!DllGetClassObject+0x94d7
contoso!DllGetClassObject+0x25ce
contoso!DllGetClassObject+0x2f7b
contoso!DllGetClassObject+0xad55
contoso!DllGetClassObject+0xaec7
contoso!DllGetClassObject+0xadf7
contoso!DllGetClassObject+0x3c00
contoso!DllGetClassObject+0x3b2a
contoso!DllGetClassObject+0x462b
USER32!UserCallWinProcCheckWow+0x13a
USER32!DispatchMessageWorker+0x1a7
contoso!DllCanUnloadNow+0x19b6
contoso!DllGetClassObject+0xeaf2
contoso+0x1d6c
litware!LitImportReportProfile+0x11c4
litware!LitImportReportProfile+0x1897
litware!LitImportReportProfile+0x1a3b
KERNEL32!BaseThreadInitThunk+0x18
ntdll!RtlUserThreadStart+0x1d
```

Ugh. A stack trace taken without working symbols. (There’s no way that `DllGetClassObject` is a deeply recursive 60KB function. Just by casual inspection, you know that the symbols are wrong.)

To see how to fix this, you just have to understand what the debugger does when it has no symbols to work from: It uses the symbols from the exported function table. For every address it wants to resolve, it looks for the nearest exported function whose address is less than or equal to the target value.

For example, suppose `CONTOSO.DLL` has the following exported symbols:

Symbol	Offset
`DllGetClassObject`	`0x5132`
`DllCanUnloadNow`	`0xFB0B`

Look at it this way: The debugger is given the following information about your module: (Diagram not to scale.)

↑	DllGetClassObject	↑	DllCanUnloadNow

It needs to assign a function to every byte in the module. In the absence of any better information, it does it like this:

???	DllGetClassObject	DllCanUnloadNow

In words, it assumes that every function begins at the location specified by the export table, and it ends one byte before the start of the next function. The debugger is trying to make the best of a bad situation.

Suppose your DLL was loaded at 0x10000000, and the debugger needs to generate a symbolic name for the address `0x1000E4F5`.

First, it converts the address into a relative virtual address by subtracting the DLL base address, leaving `0xE4F5`.

Next, it looks to see what function "contains" that address. From the algorithm described above, the debugger concludes that the address `0xE4F5` is "part of" the `DllGetClass-Object` function, which began at begins at `0x5132`. The offset into the function is therefore `0xE4F5 - 0x5132 = 0x93C3`, and it is reported in the debugger as `contoso!DllGet-ClassObject+0x93c3`.

Repeat this exercise for each address that the debugger needs to resolve, and you get the stack trace above.

Fine, now that you know how the bad symbols were generated, how do you fix it?

You fix it by undoing what the debugger did, and then redoing it with better symbols.

You need to find the better symbols. This is not too difficult if you still have a matching binary and symbol file, because you can just load up the binary into the debugger in the style of a dump file. Like Doron, you can then let the debugger do the hard work.

```
C:> ntsd -z contoso.dll
ModLoad: 10000000 10030000   contoso.dll
```

Now you just ask the debugger, "Could you disassemble this function for me?" You give it the broken symbol+offset above. The debugger looks up the symbol, applies the offset, and then *looks up the correct symbol* when disassembling.

```
0:000> u contoso!DllGetClassObject+0x93c3
contoso!CReportViewer::ActivateReport+0xe9:
10000e4f5 eb05            jmp     contoso!CReportViewer::ActivateReport+0xf0
```

Repeat for each broken symbol in the stack trace, and you have yourself a repaired stack trace.

```
litware!Ordinal3+0x6042 ← oops
litware!CViewFrame::SetInitialKeyboardFocus+0x58
litware!CViewFrame::ActivateViewInFrame+0xf2
contoso!CReportViewer::ActivateReport+0xe9
contoso!CReportViewer::LoadReport+0x12c
contoso!CReportViewer::OnConnectionCreated+0x13f
contoso!CViewer::OnConnectionEvent+0x7f
contoso!CConnectionManager::OnConnectionCreated+0x85
contoso!CReportFactory::BeginCreateConnection+0x87
contoso!CReportViewer::CreateConnectionForReport+0x20d
contoso!CViewer::CreateNewConnection+0x87
contoso!CReportViewer::CreateNewReport+0x213
contoso!CViewer::OnChangeView+0xec
contoso!CReportViewer::WndProc+0x9a7
contoso!CView::s_WndProc+0xf1
USER32!UserCallWinProcCheckWow+0x13a
USER32!DispatchMessageWorker+0x1a7
contoso!CViewer::MessageLoop+0x24e
contoso!CViewReportTask::RunViewer+0x12
contoso+0x1d6c ← oops
litware!CThreadTask::Run+0x40
litware!CThread::ThreadProc+0xe5
litware!CThread::s_ThreadProc+0x42
KERNEL32!BaseThreadInitThunk+0x18
ntdll!RtlUserThreadStart+0x1d
```

Oops, our trick doesn't work for that first entry in the stack trace, the one with `Ordinal3`. What's up with that? There is no function called `Ordinal3`!

If your module exports functions by ordinal without a name, then the debugger doesn't know what name to print for the function (since the name was stripped from the module), so it just prints the ordinal number. You will have to go back to your DLL's `DEF` file to convert the ordinal back to a function name. Or you can dump the exports from the DLL to see what functions match up with what ordinals. (Of course, for that trick to work, you need to have a matching PDB file in the symbol search path.)

In our example, suppose `litware.dll` ordinal 3 corresponds to the function `LitDebug-ReportProfile`. We would then ask the debugger

```
0:001> u litware!LitDebugReportProfile+0x6042
litware!CViewFrame::FindInitialFocusControl+0x66:
1000084f5 33db            xor     ebx,ebx
```

Okay, that takes care of our first oops. What about the second one?

In the second case, the address the debugger was asked to generate a symbol for came before the first symbol in the module. In our diagram above, it was in the area marked with question marks. The debugger has absolutely nothing to work with, so it just disassembles as relative to the start of the module.

To resolve this symbol, you take the offset and add it to the base of the module as it was loaded into the debugger, which was reported in the `ModLoad` output:

```
ModLoad: 10000000 10030000   contoso.dll
```

If that output scrolled off the screen, you can ask the debugger to show it again with the help of the `lmm` command.

```
0:001>lmm contoso*
start    end         module name
10000000 10030000   contoso    (export symbols)       contoso.dll
```

Once you have the base address, you add the offset back and ask the debugger what's there:

```
0:001> u 0x10000000+0x1d6c
contoso!CViewReportTask::Run+0x102:
100001d6c 50              push    eax
```

Okay, now that we patched up all our oopses, we have the full stack trace with symbols:

```
litware!CViewFrame::FindInitialFocusControl+0x66
litware!CViewFrame::SetInitialKeyboardFocus+0x58
litware!CViewFrame::ActivateViewInFrame+0xf2
contoso!CReportViewer::ActivateReport+0xe9
contoso!CReportViewer::LoadReport+0x12c
contoso!CReportViewer::OnConnectionCreated+0x13f
contoso!CViewer::OnConnectionEvent+0x7f
contoso!CConnectionManager::OnConnectionCreated+0x85
contoso!CReportFactory::BeginCreateConnection+0x87
contoso!CReportViewer::CreateConnectionForReport+0x20d
contoso!CViewer::CreateNewConnection+0x87
contoso!CReportViewer::CreateNewReport+0x213
contoso!CViewer::OnChangeView+0xec
contoso!CReportViewer::WndProc+0x9a7
contoso!CView::s_WndProc+0xf1
USER32!UserCallWinProcCheckWow+0x13a
USER32!DispatchMessageWorker+0x1a7
contoso!CViewer::MessageLoop+0x24e
contoso!CViewReportTask::RunViewer+0x12
contoso!CViewReportTask::Run+0x102
litware!CThreadTask::Run+0x40
litware!CThread::ThreadProc+0xe5
litware!CThread::s_ThreadProc+0x42
KERNEL32!BaseThreadInitThunk+0x18
ntdll!RtlUserThreadStart+0x1d
```

Now the fun actually starts: Figuring out why there was a break in `CViewFrame::Find-InitialFocusControl`. Happy debugging!

Bonus tip: By default, `ntsd` does not include line numbers when resolving symbols. Type `.lines` to toggle line number support.

Raymond Chen

Follow