Rescuing a stack trace the lazy way on x86-64 (aka amd64) after the debugger gave up

devblogs.microsoft.com/oldnewthing/20220526-00

May 26, 2022

Raymond Chen

Some time ago, I showed how to rescue a stack trace on x86-32 after the debugger lost the frame chain. The same basic idea works on x86-64, but it's both easy and hard, depending on how good a job you want to do.

It's hard if you want to get a perfect stack trace because Windows on x86-64 does not use the frame pointer register, so you will have to disassembly prologues to find out the size of each frame, just like we did with x86-32.

But the fact that x86-64 does not use the frame pointer register as part of the stack frame actually makes things easier if you just want to recover the trail and are okay with having a gap in coverage between where the debugger got lost and where you were able to pick things up. The fact that the code generation does not use the frame pointer for stack-walking purposes makes things easier because it's one less thing you have to recover!

So let's do things the easy way.

Here's an x86-64 stack trace that the debugger couldn't get to the bottom of:

```
Child-SP          RetAddr               Call Site
ffffb383`7a497690 fffff806`372910eb     nt!KiSwapContext+0x76
ffffb383`7a4977d0 fffff806`37292c49     nt!KiSwapThread+0x6db
ffffb383`7a4978b0 fffff806`372a8eec     nt!KiCommitThreadWait+0x159
ffffb383`7a497950 fffff806`376e13c0     nt!KeWaitForAlertByThreadId+0xc4
ffffb383`7a4979b0 fffff806`37425f75     nt!NtWaitForAlertByThreadId+0x30
ffffb383`7a4979e0 00007ff8`807c6814     nt!KiSystemServiceCopyEnd+0x25
00000000`0a02ba38 00007ff8`8075a8fd     ntdll!ZwWaitForAlertByThreadId+0x14
00000000`0a02ba40 00007ff8`1d5c6e72     ntdll!RtlAcquireSRWLockShared+0x15d
(Inline Function) --------`--------     contoso!wil::AcquireSRWLockShared+0xa
(Inline Function) --------`--------     contoso!wil::srwlock::lock_shared+0xa
(Inline Function) --------`--------
contoso!WidgetController::GetControllerInstance+0x17
00000000`0a02bab0 00007ff8`7fbe23e3     contoso!WidgetController::IsInitialized+0x32
00000000`0a02bae0 00007ff8`7fc4b68c     RPCRT4!Invoke+0x73
00000000`0a02bb30 00007ff8`7fbae5b2     RPCRT4!Ndr64StubWorker+0xb7c
00000000`0a02c1d0 00007ff8`7f5db185     RPCRT4!NdrStubCall3+0xd2
00000000`0a02c230 00007ff8`7fbc40a5     combase!CStdStubBuffer_Invoke+0x65
00000000`0a02c270 00007ff8`7f5c89cd     RPCRT4!CStdStubBuffer_Invoke+0x45
(Inline Function) --------`--------
combase!InvokeStubWithExceptionPolicyAndTracing::__l6::<lambda>::operator()+0x22
00000000`0a02c2a0 00007ff8`7f5c8767
combase!ObjectMethodExceptionHandlingAction<<lambda> >+0x4d
(Inline Function) --------`--------
combase!InvokeStubWithExceptionPolicyAndTracing+0xda
00000000`0a02c300 00007ff8`7f5de5c8     combase!DefaultStubInvoke+0x257
00000000`0a02c450 00007ff8`7f57feef     combase!SyncServerCall::StubInvoke+0x38
(Inline Function) --------`--------     combase!StubInvoke+0x496
00000000`0a02c490 00007ff8`7f55060f     combase!ServerCall::ContextInvoke+0x69f
(Inline Function) --------`--------     combase!CServerChannel::ContextInvoke+0x16b
(Inline Function) --------`--------     combase!DefaultInvokeInApartment+0x16b
00000000`0a02c770 00007ff8`7f626128     combase!ReentrantSTAInvokeInApartment+0x1df
00000000`0a02c7f0 00007ff8`7f57bfb9     combase!ComInvokeWithLockAndIPID+0x7c4
00000000`0a02cad0 00007ff8`7f577b21     combase!ThreadDispatch+0x331
00000000`0a02cbc0 00007ff8`7f31dd5c     combase!ThreadWndProc+0x1c1
00000000`0a02cc50 00007ff8`7f31c9e9     user32!UserCallWinProcCheckWow+0x33c
00000000`0a02cdc0 00007ff8`7f5df067     user32!DispatchMessageWorker+0x1c9
(Inline Function) --------`--------     combase!CCliModalLoop::MyDispatchMessage+0xc
00000000`0a02ce40 00007ff8`7f557627
combase!CCliModalLoop::PeekRPCAndDDEMessage+0x77
00000000`0a02ceb0 00007ff8`7f557366     combase!CCliModalLoop::BlockFn+0x283
00000000`0a02cf60 00007ff8`7f5e91c3     combase!ModalLoop+0xc2
00000000`0a02cfb0 00007ff8`7f550c85
combase!ClassicSTAThreadDispatchCrossApartmentCall+0x63
(Inline Function) --------`--------
combase!CSyncClientCall::SwitchAptAndDispatchCall+0x9b
00000000`0a02cff0 00007ff8`7f555372     combase!CSyncClientCall::SendReceive2+0x415
(Inline Function) --------`--------
combase!SyncClientCallRetryContext::SendReceiveWithRetry+0x2b
(Inline Function) --------`--------
combase!CSyncClientCall::SendReceiveInRetryContext+0x2f
00000000`0a02d540 00000000`00000000     combase!ClassicSTAThreadSendReceive+0xa2
```

We see that we are waiting to acquire a reader/writer lock in shared mode. This came from an inbound request from another thread, as evidenced by the fact that the entry into the Contoso component came from `RPCRT4!Invoke`, the remote procedure call runtime.

But the stack trace goes cold in the middle of the COM infrastructure. Can we try to repair it?

Start by dumping the stack at the last known good address, namely the last stack pointer before the debugger gave up:

```
00000000`0a02cff0 00007ff8`7f555372     combase!CSyncClientCall::SendReceive2+0x415

0: kd> dps 00000000`0a02cff0
00000000`0a02cff0  00000000`077801f0
00000000`0a02cff8  00000000`0a02d0f0
00000000`0a02d000  00000000`00000000
00000000`0a02d008  00000000`00000000
00000000`0a02d010  00000000`00000000
00000000`0a02d018  0000002a`00000000
00000000`0a02d020  00000000`00000000
00000000`0a02d028  00000000`00000000
00000000`0a02d030  00000000`00000000
00000000`0a02d038  00005dff`00002a78
00000000`0a02d040  4054792b`5a44dc12
00000000`0a02d048  6813c787`a838fdba
00000000`0a02d050  00000000`00000000
00000000`0a02d058  00000000`0a02dbd0
00000000`0a02d060  00000000`00000000
00000000`0a02d068  00000000`0a02d908
0: kd> d
00000000`0a02d070  00000000`0a02dbd0
00000000`0a02d078  00000000`00000000
00000000`0a02d080  0000cd1c`00000000
00000000`0a02d088  00007ff8`00000000
00000000`0a02d090  00000000`ffffffff
00000000`0a02d098  00000000`00000000
00000000`0a02d0a0  00000000`00000000
00000000`0a02d0a8  00007ff8`8074952d ntdll!RtlpLowFragHeapAllocFromContext+0x1cd
00000000`0a02d0b0  00000000`00060007
00000000`0a02d0b8  00000000`00000180
00000000`0a02d0c0  00000000`07482f80
00000000`0a02d0c8  00000000`05733dc0
00000000`0a02d0d0  00000000`074f4300
00000000`0a02d0d8  00000000`0a02d1f0
00000000`0a02d0e0  01277f18`00000000
00000000`0a02d0e8  00000000`ffffffff
```

We're looking for return addresses that look like a stack frame that would plausibly lead to `combase!ClassicSTAThreadSendReceive`. I like to play it safe and just dump for a long time, to get out of the scary broken region (whatever it was that got the debugger confused), and not pay attention until we've dumped a screenful or two.

```
(a few dump commands skipped)
0: kd> d
00000000`0a02d6f0  00000000`077801f0
00000000`0a02d6f8  00007ff8`7f5d86b6 combase!CStdIdentity::CInternalUnk::AddRef+0xc6
00000000`0a02d700  00000000`0737db01
00000000`0a02d708  00007ff8`6e2d6150 OneCoreCommonProxyStub!IID_IServiceProvider
00000000`0a02d710  00000000`074f4300
00000000`0a02d718  00000000`07366790
00000000`0a02d720  00000000`074f4300
00000000`0a02d728  00007ff8`7f57e4fd combase!StandardComClientCall::GetBuffer+0x17d
00000000`0a02d730  00000000`077801f0
00000000`0a02d738  00000000`0a02db00
00000000`0a02d740  00007ff8`00000001
00000000`0a02d748  00000000`00000000
00000000`0a02d750  00000000`00000000
00000000`0a02d758  00000000`00000000
00000000`0a02d760  00000000`0a02d7f0
00000000`0a02d768  00000000`0a02d784
0: kd> d
00000000`0a02d770  00000000`0a02d780
00000000`0a02d778  42edc8cf`91b64ada
00000000`0a02d780  00000000`b74f0a00
00000000`0a02d788  00000000`00000000
00000000`0a02d790  00000000`00000000
00000000`0a02d798  00007ff8`7f7e1bbe combase!mega__MIDL_ProcFormatString+0x340e
00000000`0a02d7a0  00000000`00000000
00000000`0a02d7a8  00000000`00000062
00000000`0a02d7b0  00000000`00000000
00000000`0a02d7b8  00000028`00000001
00000000`0a02d7c0  00007ff8`7f7a69c0 combase!IRundown_SyntaxInfo+0x50
00000000`0a02d7c8  00007ff8`7df1a950 bcryptPrimitives!GetPtrPtrToPerCpuState+0x24
00000000`0a02d7d0  00000000`00000010
00000000`0a02d7d8  00007ff8`7df1a6da bcryptPrimitives!AesRNGState_generate+0x166
00000000`0a02d7e0  11ce7436`6d5140c1
00000000`0a02d7e8  fa096000`aa003480
0: kd> d
00000000`0a02d7f0  00efe307`38512264
00000000`0a02d7f8  d800ec24`1467b218
00000000`0a02d800  00000000`00000000
00000000`0a02d808  00007ff8`7f57ced5
combase!StandardComClientCall::NegotiateSyntax+0xf5
00000000`0a02d810  0000cd1c`5d9cdd71
00000000`0a02d818  00000000`00000000
00000000`0a02d820  00000000`0a02da40
00000000`0a02d828  00000000`0737db01
00000000`0a02d830  00000000`00000000
00000000`0a02d838  00000000`0a02dbd0
00000000`0a02d840  00000000`0a02d908
00000000`0a02d848  00000000`074f4300
00000000`0a02d850  00000000`00000000
00000000`0a02d858  00007ff8`7f5789e8 combase!CClientChannel::SendReceive+0x98
```

```
00000000`0a02d860  00000000`0a02da40
00000000`0a02d868  00000000`0737db01
```

Now the `bcrypt` stuff is almost certainly spurious, probably left over from earlier calls, because there's no obvious reason for `GetPtrPtrToPerCpuState` to be calling into the COM marshaling infrastructure. However, the `combase` functions `StandardComClientCall::NegotiateSyntax` and `CClientChannel::SendReceive` look promising.

Let's tell the debugger to create a stack trace starting at a point that looks good: The parameters to the `k=` command vary from processor to processor, so I always have to go look them up.

> Kernel-Mode, x64 Processor
>
> ```
> [Processor] k[b|p|P|v] [c] [n] [f] [L] [M] [FrameCount]
> [Processor] k[b|p|P|v] [c] [n] [f] [L] [M] = StackPtr FrameCount
> [Processor] k[b|p|P|v] [c] [n] [f] [L] [M] = StackPtr InstructionPtr FrameCount
> [Processor] kd [WordCount]
> ```

The one I want here is the one that takes a stack pointer, an instruction pointer, and a frame count.

```
00000000`0a02d808  00007ff8`7f57ced5
combase!StandardComClientCall::NegotiateSyntax+0xf5
```

From this line in the stack dump, we realize that if the stack pointer is `00000000`0a02d808` and the program executes a return instruction, then control will return to `00007ff8`7f57ced5` (`combase!StandardComClientCall::NegotiateSyntax+0xf5i`) with the stack pointer equal to `00000000`0a02d810` (because returning will pop the 8-byte return address from the stack).

So we give that to the debugger and see what we get.

```
0: kd> k=00000000`0a02d810 00007ff8`7f57ced5 9
Child-SP          RetAddr               Call Site
00000000`0a02d810 00000000`0a02dcb0
combase!StandardComClientCall::NegotiateSyntax+0xf5
00000000`0a02d890 00000000`06e80730     0xa02dcb0
00000000`0a02d898 00007ff8`7f5ceb66     0x6e80730
00000000`0a02d8a0 00007ff8`7f5db368     combase!CClientChannel::GetBuffer+0x96
00000000`0a02d8d0 00007ff8`7fc4c2c3     combase!NdrExtpProxySendReceive+0x58
(Inline Function) --------`--------     RPCRT4!Ndr64pSendReceive+0x39
00000000`0a02d900 00007ff8`7fc4dd38     RPCRT4!NdrpClientCall3+0x403
00000000`0a02dc80 00007ff8`6e2548e2     RPCRT4!NdrClientCall3+0xe8
(Inline Function) --------`--------
OneCoreCommonProxyStub!IServiceProvider_RemoteQueryService_Proxy+0x2e
```

Those garbage frames at the top of the stack don't look so good, but things seem to straighten themselves out. Let's go a little further.

```
0: kd> k=00000000`0a02d810 00007ff8`7f57ced5 99
Child-SP          RetAddr               Call Site
00000000`0a02d810 00000000`0a02dcb0
combase!StandardComClientCall::NegotiateSyntax+0xf5
00000000`0a02d890 00000000`06e80730     0xa02dcb0
00000000`0a02d898 00007ff8`7f5ceb66     0x6e80730
00000000`0a02d8a0 00007ff8`7f5db368     combase!CClientChannel::GetBuffer+0x96
00000000`0a02d8d0 00007ff8`7fc4c2c3     combase!NdrExtpProxySendReceive+0x58
(Inline Function) --------`--------     RPCRT4!Ndr64pSendReceive+0x39
00000000`0a02d900 00007ff8`7fc4dd38     RPCRT4!NdrpClientCall3+0x403
00000000`0a02dc80 00007ff8`6e2548e2     RPCRT4!NdrClientCall3+0xe8
(Inline Function) --------`--------
OneCoreCommonProxyStub!IServiceProvider_RemoteQueryService_Proxy+0x2e
00000000`0a02e010 00007ff8`66d721e0
OneCoreCommonProxyStub!IServiceProvider_QueryService_Proxy+0x32
00000000`0a02e060 00007ff8`66d72085     contoso!Doodad::QueryService+0x110
00000000`0a02e0d0 00007ff8`1d5aecd7     contoso!Gadget::QueryService+0x55
00000000`0a02e100 00007ff8`1d606e6a
contoso!WidgetControllerImpl::GetWidgetDirector+0x77
(Inline Function) --------`--------
contoso!WidgetControllerImpl::UnregisterEndpoint::__l2::<lambda>::operator()+0x1a
00000000`0a02e140 00007ff8`1d5fc4d0
contoso!WidgetControllerImpl::UnregisterEndpoint+0xba
00000000`0a02e2e0 00007ff8`7fbe23e3
contoso!WidgetController::UnregisterEndpoint+0x40
00000000`0a02e310 00007ff8`7fc4b68c     RPCRT4!Invoke+0x73
00000000`0a02e360 00007ff8`7fbae5b2     RPCRT4!Ndr64StubWorker+0xb7c
00000000`0a02ea00 00007ff8`7f5db185     RPCRT4!NdrStubCall3+0xd2
00000000`0a02ea60 00007ff8`7fbc40a5     combase!CStdStubBuffer_Invoke+0x65
00000000`0a02eaa0 00007ff8`7f5c89cd     RPCRT4!CStdStubBuffer_Invoke+0x45
(Inline Function) --------`--------
combase!InvokeStubWithExceptionPolicyAndTracing::__l6::<lambda>::operator()+0x22
00000000`0a02ead0 00007ff8`7f5c8767
combase!ObjectMethodExceptionHandlingAction<<lambda> >+0x4d
(Inline Function) --------`--------
combase!InvokeStubWithExceptionPolicyAndTracing+0xda
00000000`0a02eb30 00007ff8`7f5de5c8     combase!DefaultStubInvoke+0x257
00000000`0a02ec80 00007ff8`7f57feef     combase!SyncServerCall::StubInvoke+0x38
(Inline Function) --------`--------     combase!StubInvoke+0x496
00000000`0a02ecc0 00007ff8`7f55060f     combase!ServerCall::ContextInvoke+0x69f
(Inline Function) --------`--------     combase!CServerChannel::ContextInvoke+0x16b
(Inline Function) --------`--------     combase!DefaultInvokeInApartment+0x16b
00000000`0a02efa0 00007ff8`7f626128     combase!ReentrantSTAInvokeInApartment+0x1df
00000000`0a02f020 00007ff8`7f57bfb9     combase!ComInvokeWithLockAndIPID+0x7c4
00000000`0a02f300 00007ff8`7f577b21     combase!ThreadDispatch+0x331
00000000`0a02f3f0 00007ff8`7f31dd5c     combase!ThreadWndProc+0x1c1
00000000`0a02f480 00007ff8`7f31c9e9     user32!UserCallWinProcCheckWow+0x33c
00000000`0a02f5f0 00007ff8`551912e8     user32!DispatchMessageWorker+0x1c9
00000000`0a02f670 00007ff8`55191259     contoso!Session::s_RunMessageLoop+0x80
00000000`0a02f6e0 00007ff8`7fa04e9f     contoso!Session::s_RunDirectorThread+0xe9
00000000`0a02f810 00007ff8`8072485b     KERNEL32!BaseThreadInitThunk+0x10
00000000`0a02f840 00000000`00000000     ntdll!RtlUserThreadStart+0x2b
```

This looks like a totally plausible stack trace if you ignore the instability at the top.

If you'd rather get a totally clean stack trace, then try out different starting points. From this one

```
00000000`0a02d858  00007ff8`7f5789e8 combase!CClientChannel::SendReceive+0x98
```

we get

```
0: kd>  k=00000000`0a02d860 00007ff8`7f5789e8 99
Child-SP          RetAddr               Call Site
00000000`0a02d860 00007ff8`7f5db368     combase!CClientChannel::SendReceive+0x98
00000000`0a02d8d0 00007ff8`7fc4c2c3     combase!NdrExtpProxySendReceive+0x58
(Inline Function) --------`--------     RPCRT4!Ndr64pSendReceive+0x39
00000000`0a02d900 00007ff8`7fc4dd38     RPCRT4!NdrpClientCall3+0x403
00000000`0a02dc80 00007ff8`6e2548e2     RPCRT4!NdrClientCall3+0xe8
(Inline Function) --------`--------
OneCoreCommonProxyStub!IServiceProvider_RemoteQueryService_Proxy+0x2e
00000000`0a02e010 00007ff8`66d721e0
OneCoreCommonProxyStub!IServiceProvider_QueryService_Proxy+0x32
00000000`0a02e060 00007ff8`66d72085     contoso!Doodad::QueryService+0x110
00000000`0a02e0d0 00007ff8`1d5aecd7     contoso!Gadget::QueryService+0x55
00000000`0a02e100 00007ff8`1d606e6a
contoso!WidgetControllerImpl::GetWidgetDirector+0x77
(Inline Function) --------`--------
contoso!WidgetControllerImpl::UnregisterEndpoint::__l2::<lambda>::operator()+0x1a
00000000`0a02e140 00007ff8`1d5fc4d0
contoso!WidgetControllerImpl::UnregisterEndpoint+0xba
00000000`0a02e2e0 00007ff8`7fbe23e3
contoso!WidgetController::UnregisterEndpoint+0x40
00000000`0a02e310 00007ff8`7fc4b68c     RPCRT4!Invoke+0x73
00000000`0a02e360 00007ff8`7fbae5b2     RPCRT4!Ndr64StubWorker+0xb7c
00000000`0a02ea00 00007ff8`7f5db185     RPCRT4!NdrStubCall3+0xd2
00000000`0a02ea60 00007ff8`7fbc40a5     combase!CStdStubBuffer_Invoke+0x65
00000000`0a02eaa0 00007ff8`7f5c89cd     RPCRT4!CStdStubBuffer_Invoke+0x45
(Inline Function) --------`--------
combase!InvokeStubWithExceptionPolicyAndTracing::__l6::<lambda>::operator()+0x22
00000000`0a02ead0 00007ff8`7f5c8767
combase!ObjectMethodExceptionHandlingAction<<lambda> >+0x4d
(Inline Function) --------`--------
combase!InvokeStubWithExceptionPolicyAndTracing+0xda
00000000`0a02eb30 00007ff8`7f5de5c8     combase!DefaultStubInvoke+0x257
00000000`0a02ec80 00007ff8`7f57feef     combase!SyncServerCall::StubInvoke+0x38
(Inline Function) --------`--------     combase!StubInvoke+0x496
00000000`0a02ecc0 00007ff8`7f55060f     combase!ServerCall::ContextInvoke+0x69f
(Inline Function) --------`--------     combase!CServerChannel::ContextInvoke+0x16b
(Inline Function) --------`--------     combase!DefaultInvokeInApartment+0x16b
00000000`0a02efa0 00007ff8`7f626128     combase!ReentrantSTAInvokeInApartment+0x1df
00000000`0a02f020 00007ff8`7f57bfb9     combase!ComInvokeWithLockAndIPID+0x7c4
00000000`0a02f300 00007ff8`7f577b21     combase!ThreadDispatch+0x331
00000000`0a02f3f0 00007ff8`7f31dd5c     combase!ThreadWndProc+0x1c1
00000000`0a02f480 00007ff8`7f31c9e9     user32!UserCallWinProcCheckWow+0x33c
00000000`0a02f5f0 00007ff8`551912e8     user32!DispatchMessageWorker+0x1c9
00000000`0a02f670 00007ff8`55191259     contoso!Session::s_RunMessageLoop+0x80
00000000`0a02f6e0 00007ff8`7fa04e9f     contoso!Session::s_RunDirectorThread+0xe9
00000000`0a02f810 00007ff8`8072485b     KERNEL32!BaseThreadInitThunk+0x10
00000000`0a02f840 00000000`00000000     ntdll!RtlUserThreadStart+0x2b
```

which is identical to our previous one, once you get past the initial instability.

Next time, we'll look at why there is often instability at the start of a recovered stack trace.

Raymond Chen

Follow