Adventures in application compatibility: The case of the RAII type that failed to run its destructor

devblogs.microsoft.com/oldnewthing/20220405-00

April 5, 2022

Raymond Chen

A customer reported that their system crashed with bugcheck code 0xEF: `CRITICAL_PROCESS_DIED` because the RPCSS service died. The investigation of this failure was difficult because by the time the crash occurred, everything was badly corrupted with no evidence as to who did the corrupting. The data that was being corrupted is per-thread data that is normally managed by an RAII type, so there should be no way it could be getting corrupted, assuming that the RAII type is being used properly: Always created and destructed in LIFO order, which is typically accomplished by making it a local variable in a non-coroutine function.

The team scoured their code for all uses of that RAII type and couldn't find any place where they could possibly be using it incorrectly.

Eventually, the team discovered that a third party anti-malware application had injected itself into the RPCSS service and was detouring some functions.[1] And for some reason, the detour was unhappy, but instead of failing the call, it raised a structured exception.

This structured exception was caught by the RPC equivalent of the giant try/catch that COM wraps around every server method. For RPC, this wrapper goes by the name `RpcExceptionFilter`. The exception was raised from their detour and it so happens that the exception they raised is not one that the `RpcExceptionFilter` function deems to be noteworthy, so the `RpcExceptionFilter` function swallows the exception and returns an RPC failure to the client.

As I noted, this structured exception bypasses C++ destructors.

And that's why the RAII class was not working. The code wasn't holding it wrong. Rather, the third party code broke the rules.

Now, the third party code had this problem all along: The rogue exception caused an entire RPC call to become abandoned mid-stream, resulting in lots of leaks and missing cleanup. What changed is that there's new code in the RPCSS service that was counting on the cleanup

to avoid a crash.

This particular anti-malware program must be somewhat popular because the problem recurs about once or twice a year with a different customer each time. This is the bitter spot[2] in bug frequency, because it's not frequent enough that the problem remains on your mind whenever you get a mystery crash so you can open the discussion with the customer by saying, "Okay, before I spend too much time on this, can you tell me if you recently installed or reconfigured Contoso anti-malware?" On the other hand, the issue is frequent enough that it consumes a lot of time and effort to gather crash dumps and eventually realize that it's that Contoso anti-malware problem again.

[1] Windows does not support detouring the operating system. This anti-malware software is doing unsupported things, but good luck explaining that to their customers.

[2] The bitter spot is the opposite of the sweet spot. Or is the opposite of the sweet spot the *salty spot*? The sour spot? Maybe it's all of them. When you find yourself in this spot, you are probably going to be sour, bitter and probably a little salty.

Raymond Chen

Follow