The case of the C++/WinRT cached factories that pointed into freed memory

devblogs.microsoft.com/oldnewthing/20211105-00

November 5, 2021

Raymond Chen

A customer had a program that crashed inside C++/WinRT:

```
contoso!winrt::impl::consume_Windows_ApplicationModel_AppExtensions_
        IAppExtensionCatalogStatics<winrt::Windows::ApplicationModel::
        AppExtensions::IAppExtensionCatalogStatics>::Open+0x22
contoso!winrt::Windows::ApplicationModel::AppExtensions::
        AppExtensionCatalog::Open::__l2::<lambda_...>::operator()+0x22
contoso!winrt::impl::call_factory+0x42
contoso!winrt::Windows::ApplicationModel::AppExtensions::
        AppExtensionCatalog::Open+0x4c
contoso!contoso::impl::FindContosoExtension+0xc2
contoso!ContosoSession::Initialize+0xc9
contoso!ContosoSession::Create+0xf6
contoso!ContosoSession::Run+0x34
kernel32!BaseThreadInitThunk+0x10
ntdll!RtlUserThreadStart+0x2b

contoso!winrt::impl::consume_Windows_ApplicationModel_AppExtensions_
        IAppExtensionCatalogStatics<winrt::Windows::ApplicationModel::
        AppExtensions::IAppExtensionCatalogStatics>::Open+0x22:

        mov     rax,qword ptr [rax+30h] ds:00007fff`00778030=????????????????

0:004> ln @rax
<Unloaded_AppExtension.dll>+0x28000
```

This is a call to a static method of a Windows Runtime class. We saw some time ago that static methods are implemented by the factory object. C++/WinRT caches these factory objects so that subsequent attempts to call static methods can take advantage of the work done by the first call. But here, we are trying to call the factory, only to discover that it has been unloaded!

The developers of `AllExtension.dll` say that their component follows the standard patterns, including rejecting `DllCanUnloadNow` if there are any outstanding objects. So what's going on? How could they get unloaded while there still outstanding objects?

I guessed that what happened is that somebody called `CoUninitialize`, because `CoUninitalize` will ask a DLL if it is okay to unload now, but the answer is a foregone conclusion: Whether or not the DLL says that it's okay to unload, COM is going to unload it. And that orphans the outstanding references to the DLL's factories, which are now pointers into freed memory.

Now, if your module is a DLL that exposes Windows Runtime objects, then your `DllCan-UnloadNow` is called when COM uninitializes, and the standard implementation provided by C++/WinRT empties the factory caches when this happens. That way, when COM uninitializes, all the cached factories are thrown away, seeing as they are about to become invalid.

In this case, however, `contoso.dll` does not expose any Windows Runtime objects of its own. Its use of C++/WinRT is purely as a consumer. It was not loaded by COM, and consequently, its `DllCanUnloadNow` (if it even had one) would not be called.

When the main program calls into `CreateContosoSession`, a worker thread is created to manage the Contoso session. That worker thread initializes COM when it starts and uninitializes COM when it's finished, thereby providing a courtesy to the main program, saving it the hassle of having to initialize COM.

This courtesy, however, came at great personal cost: When the Contoso session ended, the worker thread called `CoUninitialize`, thereby invalidating its own factory cache. When the host program tried to create a second session, the new worker thread crashed trying to use a factory from a no-longer-valid cache.

Part of the solution here is to remove the courtesy. Have the main program initialize COM and just leave it initialized for the lifetime of the process. Not only does that keep the C++/WinRT factory cache valid, it avoids all the wasted effort of uninitializing COM, only to initialize it again moments later.

If the host program doesn't want to dedicate a thread to keeping COM alive, it can use the `CoIncrementMTAUsage` function to keep the MTA alive.

The root cause, however, is that `contoso.dll` is itself not a COM server, so it never gets called by COM to clean up. Contoso could switch its entry point to a COM entry point (where the client uses `CoCreateInstance` or `RoActivateInstance` to load the DLL), so that COM is in control of the lifetime and will call `DllCanUnloadNow`. If that's not possible, Contoso could at least register an object in the COM static store so it can clean its factory cache when COM uninitializes.

Raymond Chen

Follow