The case of the PasswordVault.Add call that the customer thinks was hung

devblogs.microsoft.com/oldnewthing/20230407-00

April 7, 2023

Raymond Chen

A customer reported that one of their clients had a system that hung in a call to `PasswordVault.Add`. They were unable to reproduce the problem on any of their systems, but they were able to capture a Time Travel trace of the program on the client system, and they asked for our help in figuring out why it was hung and what they can do about it.

I opened the trace and found the point where they call `PasswordVaule.Add()`, then stepped over the call to see what happened.

```
contoso!winrt::impl::consume_Windows_Security_Credentials_IPasswordVault<
    winrt::Windows::Security::Credentials::IPasswordVault>::Add:
00007ff7`4332dab0 sub     rsp,28h
00007ff7`4332dab4 mov     rcx,qword ptr [rcx]
00007ff7`4332dab7 mov     rdx,qword ptr [rdx]
00007ff7`4332daba mov     rax,qword ptr [rcx]
00007ff7`4332dabd mov     rax,qword ptr [rax+30h]
00007ff7`4332dac1 call    qword ptr [contoso!__guard_dispatch_icall_fptr
(00007ff7`433f4660)]

Time Travel Position: 1B5E2B:58
contoso!winrt::impl::consume_Windows_Security_Credentials_IPasswordVault<
    winrt::Windows::Security::Credentials::IPasswordVault>::Add+0x11:
00007ff7`4332dac1 call    qword ptr [contoso!__guard_dispatch_icall_fptr
(00007ff7`433f4660)]

0:000> p
Time Travel Position: 1B69BC:98
contoso!winrt::check_hresult
    [inlined in
contoso!winrt::impl::consume_Windows_Security_Credentials_IPasswordVault<
        winrt::Windows::Security::Credentials::IPasswordVault>::Add+0x17]:
00007ff7`4332dac7 85c0            test    eax,eax
```

Hey, the call returned after all. It didn't hang.

What was the result?

```
0:000> r
rax=0000000080070425 rbx=0000000000000000 rcx=a7d3b6100ff30000
rdx=000000000000010b rsi=ffffffffffffffff rdi=000001d030897d80
rip=00007ff74332dac7 rsp=00000010d3bfca90 rbp=00000010d3bfe0a0
 r8=0000000000000001  r9=0000000000000001 r10=0000000000001410
r11=00000010d3bfca70 r12=00000010d3bfe500 r13=000001d030897d88
r14=0000000000000000 r15=000001d03020c6e0
iopl=0         nv up ei pl zr na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
contoso!winrt::check_hresult
    [inlined in
contoso!winrt::impl::consume_Windows_Security_Credentials_IPasswordVault<
        winrt::Windows::Security::Credentials::IPasswordVault>::Add+0x17]:
00007ff7`4332dac7 85c0            test    eax,eax
```

Aha, the call failed with error `0x80070425` which is the error code `ERROR_SERVICE_CANNOT_ACCEPT_CTRL` converted via `HRESULT_FROM_WIN32` to an `HRESULT`.

The call failed, and the C++/WinRT projection turns this into a C++ exception. What happened to that exception?

The customer was kind enough to share their source code, so we could continue our investigation without having to do reverse-engineering of their app.

```
namespace winrt::Contoso::implementation
{
    struct AccountPage : AccountPageT<AccountPage>
    {
        ...

        winrt::Windows::Security::Credentials::PasswordVault m_passwordVault;
    };

    IAsyncAction AccountPage::UpdatePassword()
    {
        ...
        /* Save the user's password to the password vault */
        auto cred = PasswordCredential(...);
        m_passwordVault.Add(cred);
        ...
    }

    IAsyncAction AccountPage::ConnectButton_Click(
        IInspectable const&, RoutedEventArgs const&)
    {
        // keep "this" alive while awaiting.
        auto strong_this = get_strong();

        ...

        co_await UpdatePassword();

        ...
    }
}
```

Now we can see what happens to the exception.

The exception from `m_passwordVault.Add(cred)` is not explicit caught by `UpdatePassword()` so it gets caught by the coroutine infrastructure and puts the `IAsyncAction` into a failed state. This and initiates the resumption of the awaiter, `ConnectButton_Click`.

When `ConnectButton_Click` resumes, the `co_await` asks the `IAsyncAction` what happened, and it says, "Oh, I failed. Here's the `HRESULT`." This triggers a new exception to be thrown, and since this goes uncaught by `ConnectButton_Click`, the coroutine infrastructure catches it and puts the outer `IAsyncAction` into a failed state.

But nobody is awaiting the outer `IAsyncAction`.

Windows Runtime event handlers are called when the event occurs, but the event handler returns only an `HRESULT` at the ABI layer (which is `void` at the projection layer). Although we declared our event handler as returning `IAsyncAction`, that return value is simply

discarded by the projection.

In general, the CRTP wrappers provided by C++/WinRT use duck typing: Your CRTP derived class need only provide a method which can be called with the formal parameters, and whose return value can be converted to the formal return value. It need not actually accept or return exactly those types, as long as the conversion is possible.

For example, if you are implementing a C++/WinRT method whose formal signature is

```
int64_t DoSomething(Widget widget);
```

your implementation can be

```
int32_t DoSomething(IInspectable const& object, int extra = 0);
```

The projection will do something like this:

```
int64_t retval = DoSomething(widget);
```

And that does compile. The `widget` parameter is a `Widget`, but an `IInspectable const&` can bind to a `Widget`. The extra parameter defaults to zero. And the return value can be converted from `int32_t` to `int64_t`.

You have been taking advantage of this without realizing it: Sometimes your implementation methods take a parameter const reference, and sometimes it takes it by value (if it needs to be kept alive across a suspension point).

Anyway, what happens is that the projection calls `ConnectButton_Click`, which returns an `IAsyncAction`, and the projection discards it. We saw last time that discarding an `IAsyncAction` ignores any exception that may have occurred in the coroutine.

The customer is misinterpreting an ignored exception as a hang.

If you are writing an event handler that is a coroutine, you probably should use `winrt::fire_and_forget` instead of `IAsyncAction`. If you use `IAsyncAction`, then any unhandled exceptions will just be ignored, and that's probably going to leave you scratching your head. On the other hand, `winrt::fire_and_forget` treats unhandled exceptions as fatal errors, and you'll get a crash report from your customer so you can diagnose what went wrong.

And then once you turn unhandled exceptions into fatal errors, you may decide that you want to handle those exceptions after all.

```
IAsyncAction AccountPage::UpdatePassword()
{
    ...
    /* Save the user's password to the password vault */
    auto cred = PasswordCredential(...);
    try {
        m_passwordVault.Add(cred);
    { catch (...) {
        LogFailure("Adding credentials", winrt::to_hresult());
        WarnUser("Unable to save credentials for next time");
    }
    ...
}
```

Bonus chatter: But what was causing the `ERROR_SERVICE_CANNOT_ACCEPT_CTRL` error in the first place?

One reason for this error is that the Credential Manager service could not be started. In this case, the reason was that the customer had disabled the service.

On the "Is it okay to disable this service?" page, the Credential Manager service is listed as

> Credential Manager
>
> | | |
> |---|---|
> | **Service name** | VaultSvc |
> | **Description** | Provides secure storage and retrieval of credentials to users, applications and security service packages. |
> | **Installation** | Always installed |
> | **Startup type** | Manual |
> | **Recommendation** | No guidance |
>
> Each service on the system is categorized as follows:
>
> - **Should Disable**: A security-focused enterprise will most likely prefer to disable this service and forego its functionality (see additional details below).
> - **OK to Disable**: This service provides functionality that is useful to some but not all enterprises, and security-focused enterprises that don't use it can safely disable it.
> - **Do Not Disable**: Disabling this service will impact essential functionality or prevent specific roles or features from functioning correctly. Therefore it should not be disabled.
> - **(No guidance)**: The impact of disabling these services has not been fully evaluated. Therefore, the default configuration of these services should not be changed.

(Emphasis mine.)

Bonus chatter: The customer confirmed that the client had disabled the Credential Manager service on the system. Re-enabling it fixed the problem.