When I try to call an exported function, the target crashes when it tries to call any Windows function

devblogs.microsoft.com/oldnewthing/20230922-00

September 22, 2023

Raymond Chen

A customer reported that they had a program that loaded a DLL, then called `GetProcAddress` to locate an exported function. But when they called that exported function, the DLL crashed at the point it calls out to any Windows API.

The function is exported like this:

```
extern "C" __declspec(dllexport) int AwesomeFunction();
```

but when it calls any Windows function, such as `CreateMutex`, it crashes with

```
Exception thrown at 0x0000000000025A0C in CONTOSO.exe: 0xC0000005: Access violation
executing location 0x0000000000025A0C.
```

That's not a lot of information to go on, but I guessed that they were accidentally loading their DLL with `DONT_RESOLVE_DLL_REFERENCES`.

I guessed this because the invalid code address looks a lot like a relative virtual address. Relative virtual addresses have a maximum value of the virtual size of the mapped image. If you assume that most DLLs have under 10MB of combined code and data, this puts a cap of `0x00a00000` on the relative virtual address.

For DLLs that are not bound, the entry in the import address table is the relative virtual address of the name of the function being imported. So it looks like the code is jumping not to the target function but to the relative virtual address that holds the name of the function it's trying to call. Which means that the imported function was never resolved. And that happens if you pass `DONT_RESOLVE_DLL_REFERENCES`.

The customer insisted that they weren't doing that, and they shared some code to prove it.[1]

```
// Litware, the exporting DLL

extern "C" __declspec(dllexport) int GetLitwareVersion()
{
    return 42;
}

extern "C" __declspec(dllexport) int AwesomeFunction()
{
    // If we delete this line, it just crashes at the next
    // place we call a Windows function.
    auto mutex = CreateMutexW(nullptr, FALSE, nullptr);
    if (mutex == nullptr) {
        return -1;
    }

    // ... etc ...

    CloseHandle(mutex);

    return 0;
}
```

And the consumer looks like this:

```
// Contoso, the consuming executable

#include <windows.h>

int main(int argc, char** argv)
{
    auto dll = LoadLibraryW(L"litware.exe");
    if (!dll) return 1;

    auto getLitwareVersion = ((int(*)())
        GetProcAddress(dll, "GetLitwareVersion");
    if (!getLitwareVersion) return 1;

    // This works because GetLitwareVersion doesn't
    // call any Windows functions.
    auto version = getLitwareVersion();

    auto awesomeFunction = ((int(*)())
        GetProcAddress(dll, "AwesomeFunction");
    if (!awesomeFunction) return 1;

    // This crashes once AwesomeFunction tries to call
    // a Windows function.
    auto result = awesomeFunction();

    return result;
}
```

Did you spot the problem?

The customer said they were loading a DLL, but in their code, they are loading `L"litware.exe"`. When asked about this, they said, “But it doesn’t matter. Executables can export functions too.”

While it’s true that executables can export functions, it’s not true that you can load an executable as a DLL. The documentation notes that

> If the specified module is an executable module, static imports are not loaded; instead, the module is loaded as if by `LoadLibraryEx` with the `DONT_RESOLVE_DLL_REFERENCES` flag.

So they were in fact loading the DLL with the `DONT_RESOLVE_DLL_REFERENCES` flag; they just didn’t realize it.

One lesson from this is “Don’t load executables as if they were DLLs.” But really, the purpose of this story is to show how you can diagnose a problem by looking at the evidence and combining it with what you know to come up with plausible theories as to what went wrong. Even though my theory was, strictly speaking, wrong, it steered us in the right direction.

[1] This ended up being another one of those weird psychological tricks: To get the customer to share their code, you have to accuse them of doing something silly, and that goads them into sharing the code to prove you wrong. I’m willing to suffer the disgrace of being proven wrong if it means you’ll share the malfunctioning code.