Windows is not a Microsoft Visual C/C++ Run-Time delivery channel

devblogs.microsoft.com/oldnewthing/20140411-00

April 11, 2014

Raymond Chen

There's a DLL in the system directory called `MSVCRT.DLL`, and from its name, you might think that it is the Microsoft Visual C/C++ Run-Time library. That is a perfectly reasonable guess.

But it would also be wrong.

The Microsoft Visual C/C++ Run-Time libraries go by names like `MSVCR71.DLL` or `MSVCR80.DLL` or `MSVCR90.DLL` or `MSVCR100.DLL`, and the debugging versions have a `D` in there, too. And like MFC, these binaries might be on your machine as a side effect of the implementation of a particular Windows component, but they are not contractual. If your program requires the Visual C/C++ Run-Time library, then your program needs to install the appropriate version. (There are redistributable packages you can include with your application.)

Okay, so what's with the DLL with the misleading name `MSVCRT.DLL`? The unfortunate name is a consequence of history.

Back in Windows 95, `MSVCRT.DLL` *was* the Microsoft Visual C/C++ Run-Time library, or at least it was the runtime library for Visual C/C++ 4.2. As each new version of Visual C/C++ came out, the Windows team had to go update their copy of `MSVCRT.DLL` to match. And if the Windows team wanted to fix a bug in `MSVCRT.DLL`, they had to make sure that the Visual C/C++ team made the corresponding change in their version.

This high degree of coördination became untenable, especially since it required the Windows team to do things like push a new version of `MSVCRT.DLL` to all downlevel platforms whenever a new version of Visual C/C++ came out. (Good luck doing this in the days before Windows Update!)

And sometimes these fixes caused compatibility problems. For example, I remember there was a fix for a Y2K problem which caused one application to crash because the fix altered the stack usage in such a way that exposed an uninitialized variable bug.

One serious problem with the `MVSCRT.DLL` "one runtime to rule them all" model is that multiple versions of Visual C++ would all use the same library, and keeping one DLL compatible with all versions of Visual C++ was a maintenance nightmare. For example, if a new C++ language feature required a change to the `ostream` class, you had to be careful to design your change so that the class was still binary-compatible with the older version of the class. This meant not changing the size of the class (because somebody may have derived from it) and not changing the offsets of any members, and being careful which virtual methods you call. This was in practice not done, and the result was that (for example) Windows 95 and Windows 98 both had DLLs called `MSVCRT.DLL` that were not compatible with each other.

And of course there was the problem of some application installer unwittingly overwriting the existing copy of `MSVCRT.DLL` with an older one, causing *the entire operating system* to stop working.

At some point, the decision was made to just give up and declare it an operating system DLL, to be used only by operating system components. All newer versions of Visual C/C++ used specifically-numbered DLLs for their runtime libraries. (Giving different names to each version of the run-time library solves the problem of trying to make one DLL service multiple versions of clients, as well as addressing the *accidental downgrade* problem.)

Although `MSVCRT.DLL` has been an operating system DLL for a long time, and has been documented as off-limits to applications, there are still a lot of people who treat it as a C runtime delivery channel, and those programs create a lot of grief for the product team.

I remember one change that the runtime library folks made to `MSVCRT.DLL` that had to be backed out and revisited because they found an application that not only linked to `MSVCRT.DLL` instead of the runtime library the compiler intended, but also groveled into an internal array and manipulated private members. (I was one of the people who investigated this compatibility issue, but I was not the one who solved it.)

```
// Note: The issue has been simplified for expository purposes
struct SomethingInternal
{
    int widget;
    short widgetFlags;
    char widgetLevel;
    int needs_more_time;
};
SomethingInternal InternalArray[80];
```

The runtime library folks added a new member to the structure:

```
struct SomethingInternal
{
    int widget;
    short widgetFlags;
    char widgetLevel;
    int needs_more_time;
    int needs_more_cowbell;
};
```

This change increased the size of the `SomethingInternal` structure, which in turn meant that when the application did

```
// Redeclare this internal structure in MSVCRT.DLL
// so we can poke the needs_more_time member to get more time.
struct SomethingInternal
{
    int widget;
    short widgetFlags;
    char widgetLevel;
    int needs_more_time;
};
extern SomethingInternal InternalArray[80];
...
    InternalArray[i].needs_more_time = 1;
...
```

it ended up poking the wrong byte because the structure size didn't match.

The runtime library folks had to go back and squeeze the cowbell flag into the structure in a way that didn't alter the size of the `SomethingInternal` structure. I don't remember exactly what the fix was, but one way they could've done it was by squeezing the flag into the one byte of padding between `widgetLevel` and `needs_more_time`.

```
struct SomethingInternal
{
    int widget;
    short widgetFlags;
    char widgetLevel;
    char needs_more_cowbell;
    int needs_more_time;
};
```

Bonus chatter: The application had an easy time messing with the internal array because the source code to the C runtime library is included with the compiler, So much for "All these compatibility problems would go away if you published the source code." Publishing the source code makes it *easier* to introduce compatibility problems, because it lays bare all the internal undocumented behaviors. Instead of trying to reverse-engineer the runtime library, you can just sit down and read it, and if you want to do something sneaky, you can just copy the declaration of the internal array and party on the `needs_more_time` member.

Raymond Chen

Follow