On the sadness of treating counted strings as null-terminated strings

devblogs.microsoft.com/oldnewthing/20240619-00

June 19, 2024

Raymond Chen

There are a number of data types which represent a counted string. Some of them are in the C++ standard library, like `std::string` and `std::wstring`. Some of them are Windows-specific like `BSTR` or `HSTRING`. Be careful when treating these counted strings as null-terminated strings.

Treating a counted string as a null-terminated string is a lossy operation, because any embedded nulls in the counted string are mistakenly interpreted as the end of the string.

```
std::string s = "hello\0world"s;

// This prints "hello<nul>world"
std::cout << s << std::endl;

// Copy it through c_str
std::string t = s.c_str();

// This prints "hello"
std::cout << t << std::endl;
```

The embedded null in the string `s` is treated as the string terminator when we interpret the `c_str()` as a null-terminated string, and the last part of the string is lost.

Now, you wouldn't be so silly as to copy a `std::string` that way, seeing as there is a copy constructor right there.

```
std::string t = s; // use the copy constructor
```

But when you're converting between different counted string types, you may be tempted to use the null-terminated string as the intermediary.

```
// widget.GetName() returns a winrt::hstring,
// but we want to manipulate it as a std::wstring
std::wstring name(widget.GetName().c_str());
```

Not only is there a performance penalty here, because the `std::wstring` constructor has to go look for the terminating null character, but there is also a security vulnerability: If an attacker puts an embedded null in a string, they might be able to sneak past a security check or validation.

```
bool IsAllowedName(std::wstring const& name)
{
    return name == L"alice" || name == L"bob";
}

void ProcessWidget(Widget const& widget)
{
    if (!IsAllowedName(widget.Name().c_str())) {
        throw winrt::hresult_access_denied();
    }

    ⟦ continue processing ⟧
}
```

An attacker could bypass the access check by using a widget whose name is `"alice\0haha"`, and it will be considered to have an allowed name, since the embedded null causes the `std::wstring` passed to `IsAllowedName()` to consist only of the characters leading up to the null terminator.

As another example, you might want to print a `BSTR`, which is also a counted string type, although the representation is that of a pointer to the first `wchar_t`. This means that you can often pretend that a `BSTR` is a null-terminated string, but the danger is that any embedded null will cause you to stop processing the string before you get to the end.

```
void PrintBstr(BSTR bstr)
{
    std::cout << bstr;
}
```

Sidebar: There's another danger, namely that the `BSTR` might be `nullptr`, which represents a zero-length string. However, trying to `<<` a `(wchar_t*)nullptr` will crash because the `<<` operator will dereference the null pointer while searching for the null terminator.

Okay, now that we've laid out the problem, we'll look at solutions next time.

Bonus chatter:

```
// This also prints "hello"
std::string u = "hello\0world";
std::cout << u << std::endl;
```

Bonus bonus chatter: For the specific case of converting a `winrt::hstring` to a `std::wstring`, you can just pass the `winrt::hstring` and use the `std::wstring_view` conversion constructor!

```
winrt::hstring h;
std::wstring w(h); // just construct it from the hstring
```

In our example, we would just pass the `winrt::hstring` to `IsAllowedName` and let the compiler do the conversion.

```
void ProcessWidget(Widget const& widget)
{
    if (!IsAllowedName(widget.Name())) {
        throw winrt::hresult_access_denied();
    }

    ⟦ continue processing ⟧
}
```

Bonus bonus bonus chatter: What if you are forced to produce a pointer to a null-terminated string due to some interop requirement?

In that case, you should fail the operation if the counted string contains an embedded null. For `HSTRING`, you can use the `WindowsStringHasEmbeddedNull` to check for an embedded null. The `WindowsStringHasEmbeddedNull` function caches the result, so asking a second time uses the result calculated from the first time. Mind you, scanning a string for an embedded null is probably not that expensive, so the cache doesn't buy you much, especially since you're probably about to pass the string to another function that will consume it, so the contents of the string are going to be scanned by the consumer anyway.

Arguably, the `c_str()` function should throw an exception if the counted string is not representable as a C-style null-terminated string. But what's done is done. At best, we can make up a new method name, like `safe_c_str()`?