Suspicious memory leak in std::basic_string

devblogs.microsoft.com/oldnewthing/20170209-00

February 9, 2017

Raymond Chen

A customer asked for assistance debugging a memory leak. Their memory leak detection tool was reporting a leak in the following call stack:

```
ntdll!RtlAllocateHeap
Contoso!malloc
Contoso!Allocator<unsigned short>::allocate
Contoso!std::basic_string<unsigned short,std::char_traits<unsigned
short>,Allocator<unsigned short>,_STL70>::_Copy
Contoso!std::basic_string<unsigned short,std::char_traits<unsigned
short>,Allocator<unsigned short>,_STL70>::_Grow
Contoso!std::basic_string<unsigned short,std::char_traits<unsigned
short>,Allocator<unsigned short>,_STL70>::assign
Contoso!std::basic_string<unsigned short,std::char_traits<unsigned
short>,Allocator<unsigned short>,_STL70>::assign
Contoso!std::basic_string<unsigned short,std::char_traits<unsigned
short>,Allocator<unsigned short>,_STL70>::operator=
Contoso!ConfigurationImpl::validate
```

"The `ConfigurationImpl` object itself is not being leaked. Just the string inside it."

The Visual C++ team reported that there are no known memory leaks in STL70. However, the code above is using a custom allocator, so they asked to see more of the customer's code.

And they found the smoking gun, but it wasn't in the allocator. It was in the class constructor.

```
ConfigurationImpl::ConfigurationImpl()
{
    // Initialize all members to zero.
    memset(this, 0, sizeof(ConfigurationImpl));
}
```

`basic_string`, like all STL objects, is non-POD. A POD type is roughly[1] something that can be declared in C as a plain old `struct`, such as `struct Pod { int x; int y; };`. POD types can be treated as a blob of bytes that you can manipulate with `memset`, `memcpy`, and such. Non-POD types, on the other hand, are those with things like constructors, destructors, virtual methods, all that fancy C++ stuff. You cannot treat them as just a blob of bytes because they have other fancy behaviors attached, and treating them as a blob of bytes bypasses (and may even damage) those fancy behaviors.

In this case, using `memset` to zero out a `basic_string` wipes out all the work that was performed by the `basic_string` constructor and results in the dreaded *undefined behavior*. Maybe undefined behavior manifests itself as a memory leak. Maybe it manifests itself as a crash. Maybe it manifests itself as time travel.

In practical terms, what you have there is memory corruption. When you have memory corruption, crazy things can happen. So don't corrupt memory.

The customer thanked us for our assistance and fixed their code.

[1]This is a simplified discussion, so don't haul out your language-lawyer pitchforks.

Raymond Chen

Follow