Why is my FormatMessage call crashing trying to read my insertion parameter?

devblogs.microsoft.com/oldnewthing/20131114-00

November 14, 2013

Raymond Chen

A customer was looking for assistance in debugging a crash in their product. The stack trace looked like this:

```
ntdll!_woutput_l+0x356
ntdll!_vsnwprintf_l+0x81
ntdll!_vsnwprintf+0x11
ntdll!RtlStringVPrintfWorkerW+0x3c
ntdll!RtlStringCchPrintfExW+0xa8
ntdll!RtlFormatMessageEx+0x324
KERNELBASE!BaseDllFormatMessage+0x18e
KERNELBASE!FormatMessageW+0x37
contoso!FormatWithFallbackLanguage+0x8a
contoso!RecordFailure+0x56
```

The string being formatted is `There was an error processing the object '%1'.`, and the insertion is a long (but valid) string. A unit test which passes a similarly long object name to `RecordFailure` does not crash.

What is the problem? There are clues in the stack trace.

The natural place to start is the function that calls `FormatMessage` to see what parameters are being passed in. And that's where you see something strange:

```
// Code in italics is wrong
BOOL FormatWithFallbackLanguage(
    DWORD dwMessageId, PCTSTR pszBuffer, SIZE_T cchBuffer, ...)
{
 va_list ap;
 va_start(ap, cchBuffer);
 // Format from the user's preferred language.
 DWORD cchResult = FormatMessage(
               FORMAT_MESSAGE_FROM_HMODULE,
               g_hinst, dwMessageId, g_preferredLangId,
               pszBuffer, cchBuffer, &ap);
 // If that didn't work, then use the fallback language.
 if (cchResult == 0) {
  cchResult = FormatMessage(
               FORMAT_MESSAGE_FROM_HMODULE,
               g_hinst, dwMessageId, g_fallbackLangId,
               pszBuffer, cchBuffer, &ap);
 }
 va_end(ap);
 return cchResult != 0;
}
```

(The clue in the stack trace was the word *fallback* in the function name, which suggests that if the formatting attempt fails, it'll try again some other way.)

The purpose of this function is to format the message using the specified message ID, first looking in the preferred language message table, and if that fails, by looking in the fallback language message table.

The crash occurred on the second call to `FormatMessage`. The customer said, "I'm guessing that the parameters being passed to `FormatMessage` may be causing this, but I'm not sure how to proceed. Can you provide pointers for further investigation?"

The bug is that code is reusing the `ap` parameter without resetting it. The documentation for `FormatMessage` says about the `Arguments` parameter:

> The state of the **va_list** argument is undefined upon return from the function. To use the **va_list** again, destroy the variable argument list pointer using **va_end** and reinitialize it with **va_start**.

The function `FormatWithFallbackLanguage` calls `FormatMessage`, which consumes and renders unusable the `ap` variable. If the first format attempt fails, the code passes the same (now invalid) `va_list` to a second `FormatMessage`, which is now operating on undefined data and is therefore within its rights to crash.

In practice, what happens is that the `FormatMessage` function calls `va_arg` on the `va_list` to extract the insertions, and since `va_list`s are single-use, that pretty much renders it useless for anything else. If you want to walk the parameters a second time, you need to clean up the `va_list` and then reinitialize it.

```
BOOL FormatWithFallbackLanguage(
    DWORD dwMessageId, PCTSTR pszBuffer, SIZE_T cchBuffer, ...)
{
 va_list ap;
 va_start(ap, cchBuffer);
 // Format from the user's preferred language.
 DWORD cchResult = FormatMessage(
               FORMAT_MESSAGE_FROM_HMODULE,
               g_hinst, dwMessageId, g_preferredLangId,
               pszBuffer, cchBuffer, &ap);
 // If that didn't work, then use the fallback language.
 if (cchResult == 0) {
  va_end(ap);
  va_start(ap, cchBuffer);
  cchResult = FormatMessage(
               FORMAT_MESSAGE_FROM_HMODULE,
               g_hinst, dwMessageId, g_fallbackLangId,
               pszBuffer, cchBuffer, &ap);
 }
 va_end(ap);
 return cchResult != 0;
}
```

By ending the old argument list and restarting it, the second call to `FormatMessage` has the correct starting point for extracting the variadic parameters. An alternate (and in my opinion better) way to fix the bug would be

```
BOOL FormatWithFallbackLanguage(
    DWORD dwMessageId, PCTSTR pszBuffer, SIZE_T cchBuffer, ...)
{
 va_list ap;
 // Format from the user's preferred language.
 va_start(ap, cchBuffer);
 DWORD cchResult = FormatMessage(
               FORMAT_MESSAGE_FROM_HMODULE,
               g_hinst, dwMessageId, g_preferredLangId,
               pszBuffer, cchBuffer, &ap);
 va_end(ap);
 // If that didn't work, then use the fallback language.
 if (cchResult == 0) {
  va_start(ap, cchBuffer);
  cchResult = FormatMessage(
               FORMAT_MESSAGE_FROM_HMODULE,
               g_hinst, dwMessageId, g_fallbackLangId,
               pszBuffer, cchBuffer, &ap);
  va_end(ap);
 }
 return cchResult != 0;
}
```

This avoids the "magic switcheroo" and more clearly scopes the region of validity of the `ap` variable to "solely for the purpose of the `FormatMessage` function."

Bonus chatter: Suppose the `FormatWithFallbackLanguage` accepted a `va_list` parameter directly. You might be tempted to implement it like this:

```
// code in italics is wrong
BOOL FormatWithFallbackLanguage(
    DWORD dwMessageId, PCTSTR pszBuffer, SIZE_T cchBuffer, va_list ap)
{
 va_list apOriginal = ap;
 // Format from the user's preferred language.
 DWORD cchResult = FormatMessage(
               FORMAT_MESSAGE_FROM_HMODULE,
               g_hinst, dwMessageId, g_preferredLangId,
               pszBuffer, cchBuffer, &ap);
 // If that didn't work, then use the fallback language.
 if (cchResult == 0) {
  cchResult = FormatMessage(
               FORMAT_MESSAGE_FROM_HMODULE,
               g_hinst, dwMessageId, g_fallbackLangId,
               pszBuffer, cchBuffer, &apOriginal);
 }
 return cchResult != 0;
}
```

This is not legal because a `va_list` is not directly copyable. Some architectures have rather complicated calling conventions that entail memory allocation in order to enumerate the parameters passed to variadic functions, and a bitwise copy will not respect those complexities. You have to use the `va_copy` macro to make a copy of a `va_list`.

Exercise: How did this error elude unit testing?

Exercise: What else can go wrong in this function?

Raymond Chen

Follow