Data in crash dumps are not a matter of opinion

devblogs.microsoft.com/oldnewthing/20120928-00

September 28, 2012

Raymond Chen

A customer reported a problem with the `SystemTimeToTzSpecificLocalTime` function. (Gosh, why couldn't they have reported a problem with a function with a shorter name! Now I have to type that thing over and over again.)

> We're having a problem with the `SystemTimeToTzSpecificLocalTime` function. We call it like this:
>
> ```
> s_pTimeZones->SystemTimeToTzSpecificLocalTime((BYTE)timeZoneId,
>                                  &sysTime, &localTime);
> ```
>
> On some but not all of our machines, our program crashes with the following call stack:
>
> ```
> ExceptionAddress: 77d4a0d0 (kernel32!SystemTimeToTzSpecificLocalTime+0x49)
>    ExceptionCode: c0000005 (Access violation)
>   ExceptionFlags: 00000000
> NumberParameters: 2
>    Parameter[0]: 00000000
>    Parameter[1]: 000000ac
> Attempt to read from address 000000ac
> kernel32!SystemTimeToTzSpecificLocalTime+0x49
> Contoso!CTimeZones::SystemTimeToTzSpecificLocalTime+0x26
> Contoso!CContoso::ResetTimeZone+0xc0
> Contoso!ResetTimeZoneThreadProc+0x32
> ```
>
> This problem appears to occur only with the release build; the debug build does not have this problem. Any ideas?

Notice that in the line of code the customer provided, they are *not* calling `SystemTimeToTzSpecificLocalTime`; they are instead calling some application-defined method with the same name, which takes different parameters from the system function.

The customer apologized and included the source file they were using, as well as a crash dump file.

```
void CContoso::ResetTimeZone()
{
 SYSTEMTIME sysTime, localTime;
 GetLastModifiedTime(&sysTime);
 for (int timeZoneId = 1;
      timeZoneId < MAX_TIME_ZONES;
      timeZoneId++) {
  if (!s_pTimeZones->SystemTimeToTzSpecificLocalTime((BYTE)timeZoneId,
                                &sysTime, &localTime)) {
    LOG_ERROR("...");
    return;
  }
  ... do something with localTime ...
 }
}
BOOL CTimeZones::SystemTimeToTzSpecificLocalTime(
    BYTE bTimeZoneID,
    LPSYSTEMTIME lpUniversalTime,
    LPSYSTEMTIME lpLocalTime)
{
    return ::SystemTimeToTzSpecificLocalTime(
        &m_pTimeZoneInfo[bTimeZoneID],
        lpUniversalTime, lpLocalTime);
}
```

According to the crash dump, the first parameter passed to `CTimeZones::SystemTimeToTzSpecificLocalTime` was `1`, and the `m_pTimeZoneInfo` member was `nullptr`. As a result, a bogus non-null pointer was passed as the first parameter to `SystemTimeToTzSpecificLocalTime`, which resulted in a crash when the function tried to dereference it.

This didn't require any special secret kernel knowledge; all I did was look at the stack trace and the value of the member variable.

So far, it was just a case of a lazy developer who didn't know how to debug their own code. But the reply from the customer was most strange:

> I don't think so, for two reasons.
>
> 1. The exact same build on another machine does not crash, so it must be a machine-specific or OS-specific bug.
> 2. The code in question has not changed in several months, so if the problem were in that code, we would have encountered it much earlier.

I was momentarily left speechless by this response. It sounds like the customer simply refuses to believe the information that's right there in the crash dump. "La la la, I can't hear you."

Memory values are not a matter of opinion. If you look in memory and find that the value 5 is on the stack, then the value 5 is on the stack. You can't say, "No it isn't; it's 6." You can have different opinions on how the value 5 ended up on the stack, but the fact that the value is 5 is beyond dispute.

It's like a box of cereal that has been spilled on the floor. People may argue over who spilled the cereal, or who placed the box in such a precarious position in the first place, but to take the position "There is no cereal on the floor" is a pretty audacious move.

Whether you like it or not, the value is not correct. You can't deny what's right there in the dump file. (Well, unless you think the dump file itself is incorrect.)

A colleague studied the customer's code more closely and pointed out a race condition where the thread which calls `CContoso::ResetTimeZone` may do so before the `CTimeZone` object has allocated the `m_pTimeZoneInfo` array. And it wasn't anything particularly subtle either. It went like this, in pseudocode:

```
CreateThread(ResetTimeZoneThreadProc);
s_pTimeZones = new CTimeZones;
s_pTimeZones->Initialize();
// the CTimeZones::Initialize method allocates m_pTimeZoneInfo
// among other things
```

The customer never wrote back once the bug was identified. Perhaps the sheer number of impossible things all happening at once caused their head to explode.

I discussed this incident later with another colleague, who remarked

> Frequently, some problem X will occur, and the people debugging it will say, "The only way that problem X to occur is if we are in situation Y, but we know that situation Y is impossible, so we didn't bother investigating that possibility. Can you suggest another idea?"
>
> Yeah, I can suggest another idea. "The computer is always right." You already saw that problem X occurred. If the only way that problem X can occur is if you are in situation Y, then the first thing you should do is assume that you are in situation Y and work from there."
>
> Teaching people to follow this simple axiom has avoid a lot of fruitless misdirected speculative debugging. People seem hard-wired to prefer speculation, though, and it's common to slip back into forgetting simple logic.

To put it another way:

- If X, then Y.
- X is true.
- Y cannot possible be true.

In order for these three statements to hold simultaneously, you must have found a fundamental flaw in the underlying axioms of logic as they have been understood for thousands of years.

This is unlikely to be the case.

Given that you have X right in front of you, X is true by observation. That leaves the other two statements. Maybe there's a case where X does not guarantee Y. Maybe Y is true after all.

As Sherlock Holmes is famous for saying, "When you have eliminated the impossible, whatever remains, however improbable, must be the truth." But before you rule out the impossible, make sure it's actually impossible.

Bonus chatter: Now that I've told you that the debugger never lies, I get to confuse you in a future entry by debugging a crash where the debugger lied. (Or at least wasn't telling the whole truth.)

Raymond Chen

Follow