Microspeak: Bucket bugs, bucket spray, bug spray, and failure shift

devblogs.microsoft.com/oldnewthing/20200121-00

January 21, 2020

Raymond Chen

When Windows Error Reporting receives dump files due to crashes or hangs, it tries to identify similar failures and group them together. When the number of failures of a particular type reach some level of statistical significance, it automatically files a bug against the responsible component, and the dump files are made available[1] for post-mortem analysis.

The mechanism by which this auto-grouping happens is wrapped up inside the debugger command `!analyze`, pronounced *bang analyze*. One of the things that the `!analyze` command does is produce a `FAILURE_BUCKET_ID`. This “bucket ID” is the key that is used to determine whether two issues are the same. If a lot of failures come in with the same bucket ID, then the Windows Error Reporting system automatically files a bug.

One of the major factors in getting good bugs out of Windows Error Reporting is getting a good bucket ID.

One problem is known in Microspeak as *bucket bugs*. This happens when a lot of distinct failures all wind up getting the same bucket ID. For example, when we learned about not catching exceptions you don’t intend to handle, we saw that organizing your `try` / `catch` code poorly caused every unexpected exception to get blamed on the same function, in our case, `ConvertExceptionToHResult`. As a result, `ConvertExceptionToHResult` becomes the unhappy victim of a bucket bug. The bucket ID is too coarse and fails to distinguish between unrelated failures.

The opposite problem occurs when the bucket ID is too fine and thinks that a bunch of failures are distinct when they are actually the same, resulting in *bucket spray*. A single problem manifests as a large number of different bucket IDs, producing lots of tiny little buckets with a very small number of hits (often just one). None of these buckets reach a hit count high enough to result in the automatic filing of a bug. The underlying problem may be occurring tens of thousands of times a day, but due to the bucket spray, it results in tens of thousands of one-hit buckets, and as a result, the problem eludes automatic reporting.

You can get bucket spray if the bucketing criteria include volatile information, like a GUID, a line number, or a timestamp. A module that constantly renames itself, say by incorporating the build number into the DLL name, will also result in bucket spray.

A closely-related term is *bug spray*. This sounds like the name for a can of insect repellant, but in Microspeak, bug spray is the name for the phenomenon when an automated bug-filing system files a lot of bugs that are effectively reporting the same problem, just with tiny variations.

For example, the bug-filing system may be running a series of tests over a dataset, and there is some problem with the first test that causes everything to fail. You'll wind up with a copy of the bug for each item in the dataset.

Another example could be that the bug-filing system temporarily lost access to symbols, causing all bugs to be bucketed against module offsets, which change from build to build.

The last term for today is *failure shift*. If there is a change to the system which in turn changes the bucket ID, then what happens is that the old bucket ID stops collecting hits, and all the new hits go to the new bucket ID. If you don't realize what happened, it looks like the old bug magically fixed itself, and a brand new seemingly-unrelated bug is on the rise. This can result in wasted effort, because it looks like a regression was introduced. The team tries to track down which recent change resulted in a surge in the new bucket, and then studying that change to figure out why it introduced a regression. The result of this investigation is frustration when you realize that there was no regression after all. All that happened was that an existing bug got reclassified.

[1] Subject to GDPR and other privacy regulations, of course.

Raymond Chen

Follow