STATUS_STACK_BUFFER_OVERRUN doesn't mean that there was a stack buffer overrun

devblogs.microsoft.com/oldnewthing/20190108-00

January 8, 2019

Raymond Chen

A category of dubious security vulnerability is people who recently discovered the `STATUS_STACK_BUFFER_OVERRUN` status code.

The confusion is made even worse by the fact that the human-readable description is

> The system detected an overrun of a stack-based buffer in this application. This overrun could potentially allow a malicious user to gain control of this application.

First, the history.

The `/GS` compiler switch enables the creation of a stack "cookie" value which is used as a canary to detect that a stack buffer overflow occurred. If the cookie is corrupted, then the process terminates itself immediately with the status code `STATUS_STACK_BUFFER_OVERRUN`.

It was a good name at the time it was introduced.

Over time, other reasons for immediate process termination were invented. Some of them were security-related. Others weren't. The term *fail fast* was given to these sort of *oh no things are really bad I should just stop immediately before they get any worse* immediate process termination situations.

There's even a compiler intrinsic for terminating the current process using this special status code. It seems that people can't agree on whether it's *fail fast* or *fast fail*, so in practice you see both variations.

What this means is that nowadays when you get a `STATUS_STACK_BUFFER_OVERRUN`, it doesn't actually mean that there is a stack buffer overrun. It just means that the application decided to terminate itself with great haste.

If you look in the `winnt.h` file, you'll see a list of various fast-fail codes. These codes indicate which type of fast-fail occurred.

```
#define FAST_FAIL_LEGACY_GS_VIOLATION         0
#define FAST_FAIL_VTGUARD_CHECK_FAILURE       1
#define FAST_FAIL_STACK_COOKIE_CHECK_FAILURE  2
#define FAST_FAIL_CORRUPT_LIST_ENTRY          3
#define FAST_FAIL_INCORRECT_STACK             4
#define FAST_FAIL_INVALID_ARG                 5
#define FAST_FAIL_GS_COOKIE_INIT              6
#define FAST_FAIL_FATAL_APP_EXIT              7
```

From the above list, the interesting one for me is the `FAST_FAIL_FATAL_APP_EXIT` code. This code is used when the C runtime function `abort()` gets called. And `abort()` is called by `std::terminate()`. And `std::terminate()` is called automatically for things like throwing an exception out of a `noexcept` function, or if a thrown exception goes unhandled. It's also used by some modules when an internal assertion failure is hit, or when an error occurs and the program simply wants to give up rather than try to recover. These are not necessarily security issues. It's just the program saying, "Um, I'm in trouble. I think I'll just stop right here."

Related: `STATUS_BUFFER_OVERFLOW` really should be named `STATUS_BUFFER_OVERFLOW_PREVENTED`.

Raymond Chen

Follow