Even if your assembly language code doesn't call any Windows APIs, you still have to follow the ABI

devblogs.microsoft.com/oldnewthing/20190308-00

March 8, 2019

Raymond Chen

Writing in assembly language doesn't absolve you from adhering to the software conventions. Certainly you have more flexibility in assembly language, but that doesn't give you freedom to do anything you want.

We saw earlier that you must not use any stack memory below the red zone, because a page fault or other exception could occur, and the system needs to know which parts of the stack are available for use by exception handlers.

This also means that you need to adhere to the software conventions so that the system can unwind the stack in order to find and execute the exception handlers. In particular, this means that if your function is not a lightweight leaf function, you must register unwind codes on systems that require them (which is pretty much everything other than x86).

If you fail to register unwind codes, then the system will assume that you are a lightweight leaf function, which means that it will assume that all nonvolatile registers are unmodified from the calling function, the stack pointer has not been changed from its value at function entry, and that the return address is in the default location. For x64, this means that the return address is at the top of the stack; for RISC, it means that the return address is in the standard return address register.

If the system thinks that you are a lightweight leaf function, but you didn't follow the rules for a lightweight leaf function, then bad things will happen. During unwinding, the system will not restore nonvolatile registers to the values from the calling function. (It can't do that even it wanted to, seeing as you didn't tell the system where to restore them *from*.) The system will assume that the caller's address is whatever is stored in the default return address location. (And again, it can't do any better even if it wanted to, since you didn't tell the system where the return address is.)

If the nonvolatile registers are improperly restored, then if the system needs to unwind to the caller, the caller will have incorrect registers, and that will lead to unpredictable behavior.

If the return address is improperly restored, then the system won't be able to unwind the stack at all. If you're lucky, it'll realize that the return address is not in a registered code segment and terminate your process immediately (and flag the crash as a potential security vulnerability). If you're not lucky, the incorrect return address will happen to be in a registered code segment, and the system will continue unwinding from that point, even though it has nothing to do with anything on the call stack. This may end up calling exception handlers in ways that they never expected to run, which is also something that an attacker could take advantage of.

Anyway, the point of the story is that you need to adhere to the Windows software conventions even when writing assembly language. Because the system may at times need to reason over the state of your program, and you need to keep it in a state that the system can understand.

Raymond Chen

Follow