Microspeak: Zap

devblogs.microsoft.com/oldnewthing/20100126-00

January 26, 2010

Raymond Chen

You may hear an old-timer developer use the verb *zap*.

> That proposed fix will work. Until everybody gets the fix, they can just zap the assert.

The verb *to zap* means *to replace a breakpoint instruction with an appropriate number of NOP instructions* (effectively ignoring it).

The name comes from the old Windows 2.x kernel debugger. (Actually, it may be even older, but that's as far back as I was able to trace it.) The `Z` (*zap*) command replaces the current instruction with a NOP if it is an `int 3` (the x86 single-byte breakpoint instruction), or replaced the previous instruction with NOPs if it is an `int 1` (the x86 two-byte breakpoint instruction).

This operation was quite common back in the days when lots of code was written in assembly language. A technique used by some teams was to insert a hard-coded breakpoint (called a `TRAP`) into every code path of a function. Here's an example (with comments and other identifying characteristics removed and new ones made up):

```
xyz8:   mov     bl,[eax].xyz_State
        cmp     bl,XYZSTATE_IGNORE
        TRAPe
        je      short xyz10     ; ignore this one
        or      bl,bl
        TRAPe
        je      short xyz11     ; end of table
        mov     bh,[eax].xyz_Flags
        test    bh,XYZFLAGS_HIDDEN
        TRAPz
        jz      short xyz10     ; skip - item is hidden
        test    bh,XYZFLAGS_MAGIC
        TRAPe
        je      short gvl10     ; skip - not the magic item
        TRAP
        bts     [esi].alt_flags,ALTFLAGS_SEENMAGIC
        TRAPc
        jc      short xyz10     ; weird - we shouldn't have two magic items
```

There were a variety of `TRAP` macros. Here we see the one plain vanilla `TRAP` and a bunch of fancy traps which trigger only when certain conditions are met. For example, `TRAPc` traps if the carry is set. Here's its definition:

```
TRAPc   MACRO
        local   l
        jnc     short l
        int     3
l:
        ENDM
```

Hardly rocket science.

When you became the person to trigger a particular code path for the first time, you would trigger the trap, and you either stepped through the code yourself or (if you weren't familiar with the code) contacted the author of the code to verify that the code successfully handled this "never seen before" case. When sufficiently satisfied that a code path operated as expected, the developer removed the corresponding `TRAP` from the source code.

Of course, most `TRAP`s are removed before the code gets checked in, but the ones related to error handling or recovering from data corruption tend to remain (such as here, where we inserted a `TRAP` when we encounter two magic items, which is theoretically impossible).

When you trigger one trap, you usually trigger it a lot, and you usually trigger a lot of related traps as well. The `Z` command was quite handy at neutering each one after you checked that everything was working. You zapped the trap.

That's why old-timers refer to patching out a hard-coded breakpoint as zapping, even though the *zap* command hasn't existed for over a decade.

Update: As far as I can tell, the earlier uses of the word *zap* referred to patching binaries, not for removing hard-coded breakpoints after they stopped in the debugger.

Raymond Chen

Follow