How to view the stack of threads that were terminated as part of process teardown from user mode

devblogs.microsoft.com/oldnewthing/20120518-00

May 18, 2012

Raymond Chen

Last time we saw how to view the stack of threads that were terminated as part of process teardown from the kernel debugger. You can do the same thing from a user-mode debugger, and it's actually a bit easier there. (The user-mode debugger I'm using is the one that comes with the Debugging Tools for Windows, the debugging engine that goes by a number of different front-ends, such as `ntsd`, `cdb`, and `windbg`.)

A direct translation of the kernel-mode technique from last time would involve using the `!vadump` command and picking through for the memory blocks with candidate size and attributes. But there's an easier way.

> Now would be a good point for me to remind you that this information is **for debugging purposes only**. The structures and offsets are all implementation details which can change from release to release.

Recall that the TEB begins with some pointers which bound the stack, and the seventh pointer is a self-pointer. What's even more useful is the thirteenth pointer (offset 0x30 for 32-bit TEBs, offset 0x60 for 64-bit TEBs), because that is where the PEB is stored.

Each process has a single global PEB, so all the TEBs will have the same PEB value at offset 0x30/0x60. And you can figure out the address of the current process's PEB either by using the !peb command or by simply looking at the TEB you already have.

```
0:000> dd fs:30 l1
0053:00000030  7efde000
```

Now you can search through memory looking for that value. If you see any hits at offset 0x30/0x60, then that's a candidate TEB.

The debugger normally limits memory scans to 256MB.

```
0:001> s 00000000 L 80000000 00 e0 fd 7e
                           ^ Range error in 's 00000000 l 80000000 00 e0 fd 7e'
```

Therefore, you have to issue the search eight times (for 32-bit processes) to cover the 2GB user-mode address space.

```
0:001> s 00000000 L 10000000 00 e0 fd 7e
0009e01c  00 e0 fd 7e 00 d0 fd 7e-44 e0 09 00 7b ef 17 77  ...~...~D...{..w
0009fdc0  00 e0 fd 7e 44 00 00 00-f0 ee 3a 00 10 ef 3a 00  ...~D.....:...:.
0009fe34  00 e0 fd 7e 78 fe 09 00-02 9f 18 77 00 e0 fd 7e  ...~x......w...~
0:001> s 10000000 L 10000000 00 e0 fd 7e
0:001> s 20000000 L 10000000 00 e0 fd 7e
0:001> s 30000000 L 10000000 00 e0 fd 7e
0:001> s 40000000 L 10000000 00 e0 fd 7e
0:001> s 50000000 L 10000000 00 e0 fd 7e
0:001> s 60000000 L 10000000 00 e0 fd 7e
0:001> s 70000000 L 10000000 00 e0 fd 7e
7486af70  00 e0 fd 7e 00 00 00 00-b8 00 16 77 28 00 16 77  ...~.......w(..w
7efda030  00 e0 fd 7e 00 00 00 00-00 00 00 00 00 00 00 00  ...~............
7efdd030  00 e0 fd 7e 00 00 00 00-00 00 00 00 00 00 00 00  ...~............
```

Alternatively, you can use the "length sanity check override" by inserting a question mark after the L:

```
0:001> s 00000000 L?80000000 00 e0 fd 7e
0009e01c  00 e0 fd 7e 00 d0 fd 7e-44 e0 09 00 7b ef 17 77  ...~...~D...{..w
0009fdc0  00 e0 fd 7e 44 00 00 00-f0 ee 3a 00 10 ef 3a 00  ...~D.....:...:.
0009fe34  00 e0 fd 7e 78 fe 09 00-02 9f 18 77 00 e0 fd 7e  ...~x......w...~
7486af70  00 e0 fd 7e 00 00 00 00-b8 00 16 77 28 00 16 77  ...~.......w(..w
7efda030  00 e0 fd 7e 00 00 00 00-00 00 00 00 00 00 00 00  ...~............
7efdd030  00 e0 fd 7e 00 00 00 00-00 00 00 00 00 00 00 00  ...~............
```

From the above output, we see that we can quickly reject all but the last two entries because the offset within the page is not the magic value 0x30. (This is a 32-bit process.) Hooray, two debugger commands reduce the search space to just two pages!

At this point, you can continue with the debugging technique from last time, looking at each candidate TEB to see if there's a valid stack in there.

Raymond Chen

Follow