Linux uprobe: User-Level Dynamic Tracing

28 Jun 2015

One of the features I've been looking forward to on newer Linux kernels is **uprobes**: user-level dynamic tracing, which was added to Linux 3.5 and improved in Linux 3.14. It lets you trace user-level functions; for example, the return of the readline() function from all running bash shells, with the returned string:

```
# ./uprobe 'r:bash:readline +0($retval):string'
Tracing uprobe readline (r:readline /bin/bash:0x8db60 +0($retval):string). Ctrl-C to end.
 bash-11886 [003] d... 19601837.001935: readline: (0x41e876 <- 0x48db60) arg1="ls -l"
 bash-11886 [002] d... 19601851.008409: readline: (0x41e876 <- 0x48db60) arg1="echo "hello world"
 bash-11886 [002] d... 19601854.099730: readline: (0x41e876 <- 0x48db60) arg1="df -h"
 bash-11886 [002] d... 19601858.805740: readline: (0x41e876 <- 0x48db60) arg1="cd .."
 bash-11886 [003] d... 19601898.378753: readline: (0x41e876 <- 0x48db60) arg1="foo bar"
^C
Ending tracing...
```

This is snooping on all entered commands from all shells, by instrumenting bash code dynamically. I didn't need to restart any of the bash shells to do this.

Library functions can also be traced. Here's all calls to libc sleep(), with the argument (number of seconds):

```
# ./uprobe 'p:libc:sleep %di'
Tracing uprobe sleep (p:sleep /lib/x86_64-linux-gnu/libc-2.15.so:0xbf130 %di). Ctrl-C to end.
      svscan-2134  [002] d... 19602517.962925: sleep: (0x7f2dba562130) arg1=0x5
      svscan-2134  [002] d... 19602522.963082: sleep: (0x7f2dba562130) arg1=0x5
        cron-923   [002] d... 19602524.187733: sleep: (0x7f3e26d9e130) arg1=0x3c
      svscan-2134  [002] d... 19602527.963267: sleep: (0x7f2dba562130) arg1=0x5
[...]
```

So svcscan is sleeping for 5 seconds, and cron for 60 (0x3c = 60 decimal).

uprobe is a tool I wrote for the perf-tools collection, to explore *uprobes* via Linux ftrace – the built-in tracer. (uprobe the user-level counterpart of my kprobe tool, which traces kernel functions.) uprobe is an experimental tool, and only works on newer kernels (more on this in a bit). Its real value – and why I'm blogging about it – is to show what capabilities are built into the Linux kernel. You may only ever use uprobes through another tracer (perf_events, SystemTap, LTTng, ...), and that's fine.

uprobe One-Liners

Here are the one-liners listed in the help message (-h) and man page, which summarize uprobe features:

```
# Trace readline() calls in all running "bash" executables:
uprobe p:bash:readline

# Trace readline() with explicit executable path:
uprobe p:/bin/bash:readline

# Trace the return of readline() with return value as a string:
uprobe 'r:bash:readline +0($retval):string'

# Trace sleep() calls in all running libc shared libraries:
uprobe p:libc:sleep

# Trace sleep() with register %di (x86):
uprobe 'p:libc:sleep %di'

# Trace this address (use caution: must be instruction aligned):
uprobe p:libc:0xbf130

# Trace gettimeofday() for PID 1182 only:
uprobe -p 1182 p:libc:gettimeofday

# Trace the return of fopen() only when it returns NULL:
uprobe 'r:libc:fopen file=$retval' 'file == 0'
```

Arguments to entry functions can be read from their register locations. If that's too much of a nuisance, consider Linux perf_events with debuginfo (or other tracers, like SystemTap), where the variable names can be used directly.

Warnings

I was hoping to use uprobes on the Linux 3.13 kernels I'm now debugging, but have **frequently hit issues where the target process either crashes or enters an endless spin loop**. I'm not too surprised, since uprobes is relatively new kernel code. If I'm really unlucky, the server needs a reboot, as I've wedged multiple processes. These bugs seem to have been fixed by Linux 4.0 (maybe earlier). For that reason, uprobe won't run on kernels older than 4.0 (without -F to force). Maybe that's pessimistic, and it should be 3.18 or something.

You're unlikely to need this, but I should also warn against using the raw address mode, eg "p:libc:**0xbf130**", unless you know what you are doing. uprobe is simple and does not check instruction alignment. If you use the wrong address, say, mid-way through a multi-byte instruction, the target process will either crash or be in a corrupted state. Other tools, like perf_events, use debuginfo to check instruction alignment.

Other than that, the typical warning about dynamic tracing applys: you can trace too much, slowing the target process. If that's a problem, you can try a more efficient tracer: eg, Linux perf_events, and (coming up) eBPF. You can also try uprobe with the "-d duration" option, which uses in-kernel buffering during the specified duration (at the sacrifice of live updates).

If you want to try uprobe, use a test environment first.

More Examples

See the uprobe examples file I included in perf-tools, and the man page.

uprobe Internals

The inspiration for writing uprobe came from reading the documentation in the Linux kernel, uprobetracer.txt:

```
Following example shows how to dump the instruction pointer and %ax register
at the probed text address. Probe zfree function in /bin/zsh:

    # cd /sys/kernel/debug/tracing/
    # cat /proc/`pgrep zsh`/maps | grep /bin/zsh | grep r-xp
    00400000-0048a000 r-xp 00000000 08:03 130904 /bin/zsh
    # objdump -T /bin/zsh | grep -w zfree
    0000000000446420 g    DF .text  0000000000000012  Base        zfree

  0x46420 is the offset of zfree in object /bin/zsh that is loaded at
  0x00400000. Hence the command to uprobe would be:

    # echo 'p:zfree_entry /bin/zsh:0x46420 %ip %ax' > uprobe_events

  And the same for the uretprobe would be:

    # echo 'r:zfree_exit /bin/zsh:0x46420 %ip %ax' >> uprobe_events
```

(This example uses zsh instead of bash, which might be safer when experimenting with instrumentation than doing so with the default login shell!)

These steps are a lot of mucking around (and there are additional steps: probes need clean-up, too), and beg to be automated. Fortunately, it already has been: apart from my uprobe tool, you can use these from the Linux perf command (from perf_events).

Using perf

For comparison, here's tracing using the standard Linux tracer, perf:

```
# perf probe -x /bin/bash 'readline%return +0($retval):string'
Added new event:
  probe_bash:readline  (on readline%return in /bin/bash with +0($retval):string)

You can now use it in all perf tools, such as:

    perf record -e probe_bash:readline -aR sleep 1

# perf record -e probe_bash:readline -a
^C[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.259 MB perf.data (2 samples) ]

# perf script
 bash 26239 [003] 283194.152199: probe_bash:readline: (48db60 <- 41e876) arg1="ls -l"
 bash 26239 [003] 283195.016155: probe_bash:readline: (48db60 <- 41e876) arg1="date"
# perf probe --del probe_bash:readline
Removed event: probe_bash:readline
```

The perf command provides better error checking, lower overhead, and other features; but is more cumbersome to use. I wrote uprobe to test out uprobes with ftrace, and to have a more rapid tool to explore this type of tracing. There are also some things it can do that perf can't, but more on that in another post...

UPDATE: In my next post, I covered hacking USDT with ftrace for user static probes (eg, those in Node.js, MySQL, etc), and included more examples of uprobe.

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).

About this blog

DTrace Tools
DTraceToolkit
DtkshDemos
Guessing Game
Specials
Books
Other Sites