Uprobe-tracer: Uprobe-based Event Tracing

Author: Srikar Dronamraju

Overview

Uprobe based trace events are similar to kprobe based trace events. To enable this feature, build your kernel with CONFIG_UPROBE_EVENTS=y.

Similar to the kprobe-event tracer, this doesn't need to be activated via current_tracer. Instead of that, add probe points via /sys/kernel/debug/tracing/uprobe_events, and enable it via /sys/kernel/debug/tracing/events/uprobes/<EVENT>/enable.

However unlike kprobe-event tracer, the uprobe event interface expects the user to calculate the offset of the probepoint in the object.

You can also use /sys/kernel/debug/tracing/dynamic_events instead of uprobe_events. That interface will provide unified access to other dynamic events too.

Synopsis of uprobe_tracer

```
p[:[GRP/]EVENT] PATH:OFFSET [FETCHARGS] : Set a uprobe
r[:[GRP/]EVENT] PATH:OFFSET [FETCHARGS] : Set a return uprobe (uretprobe)
p[:[GRP/]EVENT] PATH:OFFSET%return [FETCHARGS] : Set a return uprobe (uretprobe)
-:[GRP/]EVENT                           : Clear uprobe or uretprobe event

GRP           : Group name. If omitted, "uprobes" is the default value.
EVENT         : Event name. If omitted, the event name is generated based
                on PATH+OFFSET.
PATH          : Path to an executable or a library.
OFFSET        : Offset where the probe is inserted.
OFFSET%return : Offset where the return probe is inserted.

FETCHARGS     : Arguments. Each probe can have up to 128 args.
 %REG         : Fetch register REG
 @ADDR        : Fetch memory at ADDR (ADDR should be in userspace)
 @+OFFSET     : Fetch memory at OFFSET (OFFSET from same file as PATH)
 $stackN      : Fetch Nth entry of stack (N >= 0)
 $stack       : Fetch stack address.
 $retval      : Fetch return value.(\*1)
 $comm        : Fetch current task comm.
 +|-[u]OFFS(FETCHARG) : Fetch memory at FETCHARG +|- OFFS address.(\*2)(\*3)
 \IMM         : Store an immediate value to the argument.
 NAME=FETCHARG     : Set NAME as the argument name of FETCHARG.
 FETCHARG:TYPE     : Set TYPE as the type of FETCHARG. Currently, basic types
                     (u8/u16/u32/u64/s8/s16/s32/s64), hexadecimal types
                     (x8/x16/x32/x64), "string" and bitfield are supported.

(\*1) only for return probe.
(\*2) this is useful for fetching a field of data structures.
(\*3) Unlike kprobe event, "u" prefix will just be ignored, becuse uprobe
      events can access only user-space memory.
```

Types

Several types are supported for fetch-args. Uprobe tracer will access memory by given type. Prefix 's' and 'u' means those types are signed and unsigned respectively. 'x' prefix implies it is unsigned. Traced arguments are shown in decimal ('s' and 'u') or hexadecimal ('x'). Without type casting, 'x32' or 'x64' is used depends on the architecture (e.g. x86-32 uses x32, and x86-64 uses x64). String type is a special type, which fetches a "null-terminated" string from user space. Bitfield is another special type, which takes 3 parameters, bit-width, bit- offset, and container-size (usually 32). The syntax is:

```
b<bit-width>@<bit-offset>/<container-size>
```

For $comm, the default type is "string"; any other type is invalid.

Event Profiling

You can check the total number of probe hits per event via /sys/kernel/debug/tracing/uprobe_profile. The first column is the filename, the second is the event name, the third is the number of probe hits.

Usage examples

- Add a probe as a new uprobe event, write a new definition to uprobe_events as below (sets a uprobe at an offset of 0x4245c0 in the executable /bin/bash):

  ```
  echo 'p /bin/bash:0x4245c0' > /sys/kernel/debug/tracing/uprobe_events
  ```

- Add a probe as a new uretprobe event:

  ```
  echo 'r /bin/bash:0x4245c0' > /sys/kernel/debug/tracing/uprobe_events
  ```

- Unset registered event:

  ```
  echo '-:p_bash_0x4245c0' >> /sys/kernel/debug/tracing/uprobe_events
  ```

- Print out the events that are registered:

  ```
  cat /sys/kernel/debug/tracing/uprobe_events
  ```

- Clear all events:

  ```
  echo > /sys/kernel/debug/tracing/uprobe_events
  ```

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
```

> **Note**
>
> User has to explicitly calculate the offset of the probe-point in the object.

We can see the events that are registered by looking at the uprobe_events file.

```
# cat uprobe_events
p:uprobes/zfree_entry /bin/zsh:0x00046420 arg1=%ip arg2=%ax
r:uprobes/zfree_exit /bin/zsh:0x00046420 arg1=%ip arg2=%ax
```

Format of events can be seen by viewing the file events/uprobes/zfree_entry/format.

```
# cat events/uprobes/zfree_entry/format
name: zfree_entry
ID: 922
format:
     field:unsigned short common_type;         offset:0;  size:2; signed:0;
     field:unsigned char common_flags;         offset:2;  size:1; signed:0;
     field:unsigned char common_preempt_count; offset:3;  size:1; signed:0;
     field:int common_pid;                     offset:4;  size:4; signed:1;
     field:int common_padding;                 offset:8;  size:4; signed:1;

     field:unsigned long __probe_ip;           offset:12; size:4; signed:0;
     field:u32 arg1;                           offset:16; size:4; signed:0;
     field:u32 arg2;                           offset:20; size:4; signed:0;

print fmt: "(%lx) arg1=%lx arg2=%lx", REC->__probe_ip, REC->arg1, REC->arg2
```

Right after definition, each event is disabled by default. For tracing these events, you need to enable it by:

```
# echo 1 > events/uprobes/enable
```

Lets start tracing, sleep for some time and stop tracing.

```
# echo 1 > tracing_on
# sleep 20
# echo 0 > tracing_on
```

Also, you can disable the event by:

```
# echo 0 > events/uprobes/enable
```

And you can see the traced information via /sys/kernel/debug/tracing/trace.

```
# cat trace
# tracer: nop
#
#           TASK-PID    CPU#    TIMESTAMP  FUNCTION
#              | |       |          |         |
              zsh-24842 [006] 258544.995456: zfree_entry: (0x446420) arg1=446420 arg2=79
              zsh-24842 [007] 258545.000270: zfree_exit:  (0x446540 <- 0x446420) arg1=446540 arg2=0
              zsh-24842 [002] 258545.043929: zfree_entry: (0x446420) arg1=446420 arg2=79
              zsh-24842 [004] 258547.046129: zfree_exit:  (0x446540 <- 0x446420) arg1=446540 arg2=0
```

Output shows us uprobe was triggered for a pid 24842 with ip being 0x446420 and contents of ax register being 79. And uretprobe was triggered with ip at 0x446540 with counterpart function entry at 0x446420.