author	Marco Elver <elver@google.com>	2022-10-31 10:35:13 +0100
committer	Greg Kroah-Hartman <gregkh@linuxfoundation.org>	2022-11-26 09:27:52 +0100
commit	b09221f1b4944d2866d06ac35e59d7a6f8916c9f (patch)	
tree	a5e7806a2e1430479d615ab294734710d26cdd8c	
parent	f59a5ffaf9d969ef0b139a94072706c2880724d3 (diff)	
download	linux-b09221f1b4944d2866d06ac35e59d7a6f8916c9f.tar.gz	

perf: Improve missing SIGTRAP checking

[Upstream commit bb88f9695460bec25aa30ba9072595025cf6c8af]

To catch missing SIGTRAP we employ a WARN in __perf_event_overflow(),
which fires if pending_sigtrap was already set: returning to user space
without consuming pending_sigtrap, and then having the event fire again
would re-enter the kernel and trigger the WARN.

This, however, seemed to miss the case where some events not associated
with progress in the user space task can fire and the interrupt handler
runs before the IRQ work meant to consume pending_sigtrap (and generate
the SIGTRAP).

syzbot gifted us this stack trace:

```
 | WARNING: CPU: 0 PID: 3607 at kernel/events/core.c:9313 __perf_event_overflow
 | Modules linked in:
 | CPU: 0 PID: 3607 Comm: syz-executor100 Not tainted 6.1.0-rc2-syzkaller-00073-g88619e77b33d #0
 | Hardware name: Google Google Compute Engine/Google Compute Engine, BIOS Google 10/11/2022
 | RIP: 0010:__perf_event_overflow+0x498/0x540 kernel/events/core.c:9313
 | <...>
 | Call Trace:
 |  <TASK>
 |  perf_swevent_hrtimer+0x34f/0x3c0 kernel/events/core.c:10729
 |  __run_hrtimer kernel/time/hrtimer.c:1685 [inline]
 |  __hrtimer_run_queues+0x1c6/0xfb0 kernel/time/hrtimer.c:1749
 |  hrtimer_interrupt+0x31c/0x790 kernel/time/hrtimer.c:1811
 |  local_apic_timer_interrupt arch/x86/kernel/apic/apic.c:1096 [inline]
 |  __sysvec_apic_timer_interrupt+0x17c/0x640 arch/x86/kernel/apic/apic.c:1113
 |  sysvec_apic_timer_interrupt+0x40/0xc0 arch/x86/kernel/apic/apic.c:1107
 |  asm_sysvec_apic_timer_interrupt+0x16/0x20 arch/x86/include/asm/idtentry.h:649
 | <...>
 |  </TASK>
```

In this case, syzbot produced a program with event type
PERF_TYPE_SOFTWARE and config PERF_COUNT_SW_CPU_CLOCK. The hrtimer
manages to fire again before the IRQ work got a chance to run, all while
never having returned to user space.

Improve the WARN to check for real progress in user space: approximate
this by storing a 32-bit hash of the current IP into pending_sigtrap,
and if an event fires while pending_sigtrap still matches the previous

IP, we assume no progress (false negatives are possible given we could
return to user space and trigger again on the same IP).

Fixes: ca6c21327c6a ("perf: Fix missing SIGTRAPs")
Reported-by: syzbot+b8ded3e2e2c6adde4990@syzkaller.appspotmail.com
Signed-off-by: Marco Elver <elver@google.com>
Signed-off-by: Peter Zijlstra (Intel) <peterz@infradead.org>
Link: https://lkml.kernel.org/r/20221031093513.3032814-1-elver@google.com
Signed-off-by: Sasha Levin <sashal@kernel.org>

Diffstat

-rw-r--r-- kernel/events/core.c 25

1 files changed, 19 insertions, 6 deletions

```
diff --git a/kernel/events/core.c b/kernel/events/core.c
index 072ab26269c0b3..bec18d81b11611 100644
--- a/kernel/events/core.c
+++ b/kernel/events/core.c
@@ -9282,14 +9282,27 @@ static int __perf_event_overflow(struct perf_event *event,
        }

        if (event->attr.sigtrap) {
-               /*
-                * Should not be able to return to user space without processing
-                * pending_sigtrap (kernel events can overflow multiple times).
-                */
-               WARN_ON_ONCE(event->pending_sigtrap && event->attr.exclude_kernel);
+               unsigned int pending_id = 1;
+
+               if (regs)
+                       pending_id = hash32_ptr((void *)instruction_pointer(regs)) ?: 1;
                if (!event->pending_sigtrap) {
-                       event->pending_sigtrap = 1;
+                       event->pending_sigtrap = pending_id;
                        local_inc(&event->ctx->nr_pending);
+               } else if (event->attr.exclude_kernel) {
+                       /*
+                        * Should not be able to return to user space without
+                        * consuming pending_sigtrap; with exceptions:
+                        *
+                        *  1. Where !exclude_kernel, events can overflow again
+                        *     in the kernel without returning to user space.
+                        *
+                        *  2. Events that can overflow again before the IRQ-
+                        *     work without user space progress (e.g. hrtimer).
+                        *     To approximate progress (with false negatives),
+                        *     check 32-bit hash of the current IP.
+                        */
+                       WARN_ON_ONCE(event->pending_sigtrap != pending_id);
                }
                event->pending_addr = data->addr;
                irq_work_queue(&event->pending_irq);
```