author	Jiri Olsa <jolsa@kernel.org>	2022-09-16 09:19:14 +0200
committer	Greg Kroah-Hartman <gregkh@linuxfoundation.org>	2022-11-26 09:27:56 +0100
commit	2e5399879024fedd6cdc41f73fbf9bbe7208f899 (patch)	
tree	813a96ac3f3e21ec8f1c87db8f37491cc4bdf07a	
parent	717b9b4f38703d7f5293059e3a242d16f76fa045 (diff)	
download	linux-2e5399879024fedd6cdc41f73fbf9bbe7208f899.tar.gz	

diff options

context: 3
space: include
mode: unified

bpf: Prevent bpf program recursion for raw tracepoint probes

```
commit 05b24ff9b2cfabfcfd951daaa915a036ab53c9e1 upstream.

We got report from sysbot [1] about warnings that were caused by
bpf program attached to contention_begin raw tracepoint triggering
the same tracepoint by using bpf_trace_printk helper that takes
trace_printk_lock lock.

 Call Trace:
  <TASK>
  ? trace_event_raw_event_bpf_trace_printk+0x5f/0x90
  bpf_trace_printk+0x2b/0xe0
  bpf_prog_a9aec6167c091eef_prog+0x1f/0x24
  bpf_trace_run2+0x26/0x90
  native_queued_spin_lock_slowpath+0x1c6/0x2b0
  _raw_spin_lock_irqsave+0x44/0x50
  bpf_trace_printk+0x3f/0xe0
  bpf_prog_a9aec6167c091eef_prog+0x1f/0x24
  bpf_trace_run2+0x26/0x90
  native_queued_spin_lock_slowpath+0x1c6/0x2b0
  _raw_spin_lock_irqsave+0x44/0x50
  bpf_trace_printk+0x3f/0xe0
  bpf_prog_a9aec6167c091eef_prog+0x1f/0x24
  bpf_trace_run2+0x26/0x90
  native_queued_spin_lock_slowpath+0x1c6/0x2b0
  _raw_spin_lock_irqsave+0x44/0x50
  bpf_trace_printk+0x3f/0xe0
  bpf_prog_a9aec6167c091eef_prog+0x1f/0x24
  bpf_trace_run2+0x26/0x90
  native_queued_spin_lock_slowpath+0x1c6/0x2b0
  _raw_spin_lock_irqsave+0x44/0x50
  __unfreeze_partials+0x5b/0x160
  ...

The can be reproduced by attaching bpf program as raw tracepoint on
contention_begin tracepoint. The bpf prog calls bpf_trace_printk
helper. Then by running perf bench the spin lock code is forced to
take slow path and call contention_begin tracepoint.

Fixing this by skipping execution of the bpf program if it's
already running, Using bpf prog 'active' field, which is being
currently used by trampoline programs for the same reason.

Moving bpf_prog_inc_misses_counter to syscall.c because
```

trampoline.c is compiled in just for CONFIG_BPF_JIT option.

Reviewed-by: Stanislav Fomichev <sdf@google.com>
Reported-by: syzbot+2251879aa068ad9c960d@syzkaller.appspotmail.com
[1] https://lore.kernel.org/bpf/YxhFe3EwqchC%2FfYf@krava/T/#t
Signed-off-by: Jiri Olsa <jolsa@kernel.org>
Link: https://lore.kernel.org/r/20220916071914.7156-1-jolsa@kernel.org
Signed-off-by: Alexei Starovoitov <ast@kernel.org>
Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>

Diffstat

-rw-r--r--	include/linux/bpf.h	4
-rw-r--r--	kernel/bpf/syscall.c	11
-rw-r--r--	kernel/bpf/trampoline.c	15
-rw-r--r--	kernel/trace/bpf_trace.c	6

4 files changed, 23 insertions, 13 deletions

```
diff --git a/include/linux/bpf.h b/include/linux/bpf.h
index 80fc8a88c610da..73662fbabd78fa 100644
--- a/include/linux/bpf.h
+++ b/include/linux/bpf.h
@@ -1967,6 +1967,7 @@ static inline bool unprivileged_ebpf_enabled(void)
        return !sysctl_unprivileged_bpf_disabled;
 }

+void notrace bpf_prog_inc_misses_counter(struct bpf_prog *prog);
 #else /* !CONFIG_BPF_SYSCALL */
 static inline struct bpf_prog *bpf_prog_get(u32 ufd)
 {
@@ -2176,6 +2177,9 @@ static inline bool unprivileged_ebpf_enabled(void)
        return false;
 }

+static inline void bpf_prog_inc_misses_counter(struct bpf_prog *prog)
+{
+}
 #endif /* CONFIG_BPF_SYSCALL */

 void __bpf_free_used_btfs(struct bpf_prog_aux *aux,
```

```
diff --git a/kernel/bpf/syscall.c b/kernel/bpf/syscall.c
index 22e7a805c67239..0e758911d963fc 100644
--- a/kernel/bpf/syscall.c
+++ b/kernel/bpf/syscall.c
@@ -2094,6 +2094,17 @@ struct bpf_prog_kstats {
        u64 misses;
 };

+void notrace bpf_prog_inc_misses_counter(struct bpf_prog *prog)
+{
+       struct bpf_prog_stats *stats;
+       unsigned int flags;
+
+       stats = this_cpu_ptr(prog->stats);
+       flags = u64_stats_update_begin_irqsave(&stats->syncp);
+       u64_stats_inc(&stats->misses);
+       u64_stats_update_end_irqrestore(&stats->syncp, flags);
+}
+
 static void bpf_prog_get_stats(const struct bpf_prog *prog,
                               struct bpf_prog_kstats *stats)
 {
```

```
diff --git a/kernel/bpf/trampoline.c b/kernel/bpf/trampoline.c
index ad76940b02ccf8..41b67eb83ab3f3 100644
--- a/kernel/bpf/trampoline.c
+++ b/kernel/bpf/trampoline.c
@@ -863,17 +863,6 @@ static __always_inline u64 notrace bpf_prog_start_time(void)
        return start;
 }

-static void notrace inc_misses_counter(struct bpf_prog *prog)
-{
-       struct bpf_prog_stats *stats;
-       unsigned int flags;
-
-       stats = this_cpu_ptr(prog->stats);
-       flags = u64_stats_update_begin_irqsave(&stats->syncp);
-       u64_stats_inc(&stats->misses);
-       u64_stats_update_end_irqrestore(&stats->syncp, flags);
-}
-
 /* The logic is similar to bpf_prog_run(), but with an explicit
  * rcu_read_lock() and migrate_disable() which are required
  * for the trampoline. The macro is split into
@@ -896,7 +885,7 @@ u64 notrace __bpf_prog_enter(struct bpf_prog *prog, struct bpf_tramp_run_ctx *ru
        run_ctx->saved_run_ctx = bpf_set_run_ctx(&run_ctx->run_ctx);

        if (unlikely(this_cpu_inc_return(*(prog->active)) != 1)) {
-               inc_misses_counter(prog);
+               bpf_prog_inc_misses_counter(prog);
                return 0;
        }
        return bpf_prog_start_time();
@@ -967,7 +956,7 @@ u64 notrace __bpf_prog_enter_sleepable(struct bpf_prog *prog, struct bpf_tramp_r
        might_fault();

        if (unlikely(this_cpu_inc_return(*(prog->active)) != 1)) {
-               inc_misses_counter(prog);
+               bpf_prog_inc_misses_counter(prog);
                return 0;
        }


diff --git a/kernel/trace/bpf_trace.c b/kernel/trace/bpf_trace.c
index b1daf7c9b895ac..ec4b81007796c2 100644
--- a/kernel/trace/bpf_trace.c
+++ b/kernel/trace/bpf_trace.c
@@ -2058,9 +2058,15 @@ static __always_inline
 void __bpf_trace_run(struct bpf_prog *prog, u64 *args)
 {
        cant_sleep();
+       if (unlikely(this_cpu_inc_return(*(prog->active)) != 1)) {
+               bpf_prog_inc_misses_counter(prog);
+               goto out;
+       }
        rcu_read_lock();
        (void) bpf_prog_run(prog, args);
        rcu_read_unlock();
+out:
+       this_cpu_dec(*(prog->active));
 }

 #define UNPACK(...)                    __VA_ARGS__
```