perf tools + BPF

LSF/MM/BPF 2024

Namhyung Kim

BPF usage in the perf tools

- perf stat --bpf-counters
- perf trace
- perf record (--off-cpu and --filter)
- perf lock contention
- perf ftrace latency
- perf kwork

BPF sample filter

- perf record + event filter
  - whether it drops the sample
    - based on the return value of BPF program
    - BPF_PROG_TYPE_PERF_EVENT
  - BPF skeleton + map (for filter data)
    - don’t compile BPF program for each filter

sample filter example

```
$ perf record -e cycles:u --filter 'tid == 1234' myprog
```

```
for (i = 0; i < MAX_FILTERS; i++) {
    entry = bpf_map_lookup_elem(&filters, i);
    if (entry == NULL) break;
    switch (entry->op) {
    case PBF_OP_EQ:
        if (!(entry->value == ctx->sample->value))
            return 0;
    ...
```

```
struct perf_bpf_filter_entry {
    .op = PBF_OP_EQ,
    .flags = SAMPLE_TYPE_TID,
    .value = 1234,
};
```

filters (MAP_TYPE_ARRAY)

in tools/perf/util/bpf_skel/sample_filter.bpf.c:

Unprivileged BPF for perf_event

- perf event requires
  - CAP_PERFMON or
  - /proc/sys/kernel/perf_event_paranoid
- do we need CAP_BPF too?
  - for normal users who profile their own processes
  - what if BPF is allowed to access sample data only?

What can we do?

- BPF token
- pin on BPF-fs
- new unprivileged PROG_TYPE
  - PROG_LOAD, MAP_CREATE
  - bpf_map_lookup_elem(), bpf_cast_to_kern_ctx()
- or else?

Stacktrace issues

- skip BPF callstacks
- task callstack + stack-id
- deferred user callstack

Skipping BPF call stacks

- To get to the interesting part directly
  - how many entries to skip?

```
# perf lock contention -abv --stack-skip 0 -- sleep 1

[SKIP]
   0xffffffffc00a4c2c
   0xffffffffc00a4c2c
   0xffffffffaf64117e  bpf_trace_run2+0x8e
   0xffffffffb0011769  __mutex_lock.constprop.0+0x199
   0xffffffffaf85936d  do_epoll_wait+0x23d
   0xffffffffaf8598fb  do_epoll_pwait.part.0+0xb
   0xffffffffaf85aff5  __x64_sys_epoll_pwait+0x95
   0xffffffffafffa46d  do_syscall_64+0x5d
```

Section for the BPF code

- Like sched and lock functions

in arch/x86/kernel/vmlinux.lds.S:

```
.text : ... {
    _text = .
    _stext = .
    /* bootstrapping code */
    HEAD_TEXT
    TEXT_TEXT
    SCHED_TEXT
    LOCK_TEXT
    KPROBES_TEXT
    SOFTIRQENTRY_TEXT
    ...
```

in include/asm-generic/vmlinux.lds.h:

```
#define SCHED_TEXT                \
   ALIGN_FUNCTION()               \
   __sched_text_start = .;        \
   *(.sched.text)                 \
   __sched_text_end = .;
```

New flags for the stack helpers

- bpf_get_stackid(ctx, map, flags)
  - flags = BPF_F_… | <# skip>
  - BPF_F_SKIP_BPF_FN
  - BPF_F_SKIP_SCHED_FN
  - BPF_F_SKIP_LOCK_FN

Stack trace for other task

- BPF helpers
  - bpf_get_stack()
  - bpf_get_stackid()
  - bpf_get_task_stack()
- Can we add **bpf_get_task_stackid()** too?
  - for perf lock contention
  - to track stack trace of mutex owners

Deferred user stack trace

- split kernel and user stack trace
  - collect user stack when returning to user
  - better to collect build-ID and offset
  - from S-Frame work
- BPF support?
  - how to connect them

Symbolizing locks

- lock addresses need to be symbolized
  - global locks are ok (kallsyms)
  - what about others? (inode, vma, …)
- Can BTF help?
  - per-cpu lock (rq)
  - per-process lock (mmap_lock)
  - iterators?
- Data type profiling?

Symbolizing locks with BTF

- a specific use case of data type profiling
- find the caller (from callstack)
  - find the type of the first argument of lock function
    - ex) mutex_lock(), down_read(), …
    - &mm_struct.mmap_lock, &inode->i_lock, …
- track types from function arg (in BTF)
  - global variables
  - local variables ?

Summary

- Unprivileged BPF for sample filter
- Stack trace
  - skip BPF callstacks
  - bpf_get_task_stackid()
  - deferred user callstack support
- Data type profiling
  - global variable support in BTF