Debug with BPF in a Container

Song Liu, LSFMMBPF 2022

BPF for debugging

- BPF enables great debugging tools, as long as you have CAP_BPF
- CAP_BPF is not really secure
- Not friendly to containers
- Is there a sweet spot that is secure and useful?

Secure BPF programs

- Tool writers define secure BPF progs for non-root users (setcap, pinned programs/maps). cons: not flexible.
- **Mandatory filtering based on ownership**
    - Non-root user’s BPF program only triggers on events owned by this user
    - Ownership: current task, perf_event, socket, etc.
- Security enhanced map. For example, task local storage, non-root user can only access elements on its own tasks

Survey of 41 tools in bcc/libbpf-tools

- 24 tools use filtering on current task
- 3 tools with "start-end" model
  - Start-prog adds a value to hash map (process context)
  - End-prog calculates the output (irq context)
  - Example: I/O latency
- 4 tools use sched_switch or similar tracepoints
- 3 tools attach to perf_event,
- 4 tools work with socket (tcp_connect, etc.)
- 3 tools on tracepoint/kprobe/fentry in IRQ, softirq

Ownership: current and perf_event

- Filtering on current task is good for process context
- Non-root user can create task perf_event

Ownership: current and perf_event

- 24 tools use filtering on current task 👍
- 3 tools with "start-end" model
- 4 tools uses sched_switch or similar tracepoints
- 3 tools attach to perf_event 👍
- 4 tools work with socket (tcp_connect, etc.)
- 3 tools on tracepoint/kprobe/fentry in IRQ, softirq

Handle start-end model

- Start-prog is filtered based on current task
- Add a key (bio, skb, etc.) to a hash (BTF enabled? Referenced?) map in start-prog
- Filter before end-prog, if the key is not in the hash map, skip end-prog. Remove key after end-prog

Ownership: start-end model

- 24 tools use filtering on current task 👍
- 3 tools with "start-end" model 👍
- 4 tools uses sched_switch or similar tracepoints
- 3 tools attach to perf_event 👍
- 4 tools work with socket (tcp_connect, etc.)
- 3 tools on tracepoint/kprobe/fentry in IRQ, softirq

Handle Socket

- A root-maintained program that manages maps of socked owned by a user
- Filter based on key in this map

Ownership: socket

- 24 tools use filtering on current task 👍
- 3 tools with "start-end" model 👍
- 4 tools uses sched_switch or similar tracepoints
- 3 tools attach to perf_event 👍
- 4 tools work with socket (tcp_connect, etc.) 👍
- 3 tools on tracepoint/kprobe/fentry in IRQ, softirq

Handle sched_switch

- Worth some special handling
- Use two programs, one for prev task, the other for next task
- Apply filtering based prev or next

Ownership: sched_switch

- 24 tools use filtering on current task 👍
- 3 tools with "start-end" model 👍
- 4 tools uses sched_switch or similar tracepoints 👍
- 3 tools attach to perf_event 👍
- 4 tools work with socket (tcp_connect, etc.) 👍
- 3 tools on tracepoint/kprobe/fentry in IRQ, softirq

38/41 (92.7%) useful

- 24 tools use filtering on current task 👍
- 3 tools with "start-end" model 👍
- 4 tools uses sched_switch or similar tracepoints 👍
- 3 tools attach to perf_event 👍
- 4 tools work with socket (tcp_connect, etc.) 👍
- 3 tools on tracepoint/kprobe/fentry in IRQ, softirq

Use trampoline for filtering

- Root (or kernel) controlled fentry program(s) attached to each non-root BPF program to decide whether the non-root BPF program should run
- Default skip mode, if no fentry program is presented, skip the non-root BPF program

bpf_lsm_bpf

- Hook(s) on sys_bpf(). Root attaches programs to decide whether an operation is allowed for non-root users.
- Verifier generates attributes about the program, which will be used to decide whether BPF_PROG_ATTACH is allowed. Potential attributes:
  - Uses sensitive helpers (e.g., bpf_probe_read_kernel)
  - Needs filtering based on socket
  - etc.

Key kernel work, TBDs

- Default skip mode
  - How to attach the fentry program?
- bpf_lsm_bpf hook(s)
- Verifier generates attributes of a program