BPF Conformance

Testing against the IETF BPF ISA specification

Goals

- Overview of the BPF conformance suite
- Questions this presentation hopes to answer:
  - Why test?
  - What is being tested?
  - How is it being tested?
  - Who is using it?
  - What should be tested that isn’t currently being tested?

Conformance

- The behavior of a runtime when executing BPF instructions
- Asks the question: Does the runtime implement the ISA correctly?
- Ensure fidelity of the execution to the developer’s intent
- Verifier safety – Incorrect implementation can lead to security problems
- Developer confidence that programs will execute as expected
- Validates the BPF ISA specification against the Linux JIT

What is being measured

- Check a runtime’s implementation of specific BPF instructions
- Tests if all instructions in a conformance group are implemented
- Tests for common implementation errors (sign extend as an example)
- Derived from the uBPF self-tests
- Slowly being extended as missing cases are discovered

How conformance is measure

- Each tests has three parts
- Pre-invariant
    - Initial register state
    - Initial stack contents
    - Initial memory contents
- Code to execute
- Post-invariant
    - Currently only tests r0 against an expected value
    - Might be expanded further over time

Projects using bpf_conformance

- https://github.com/Alan-Jowett/bpf_conformance
- uBPF – The project where this originated
- eBPF-For-Windows
- Prevail – Used to check the verifier's model of the instructions
- rbpf – Rust-based BPF runtime

Establishing a conformance baseline

- Linux BPF implementations is the de facto standard
- Tests are executed against Linux
- If the test fails, it’s a bug
  - Test bug (common)
  - BPF ISA bug (rare)
  - Linux kernel bug (not found yet)
- Permits black-box observation of the Linux Kernel’s behavior
  - Required to preserve licensing of other projects

Example test

- # Copyright (c) Big Switch Networks, Inc
- # SPDX-License-Identifier: Apache-2.0
- -- asm
- ldxh %r0, [%r1]
- be16 %r0
- exit
- -- mem
- 11 22
- -- result
- 0x1122

What should the conformance tests measure?

- Checks for invalid instruction sequences?
- Invalid instruction sequence may become valid
- Test for psABI?
  - If so, which one?
- Currently only r0 is measured on exit
  - Is this sufficient?
  - Should this include additional state?
  - Should it include number of instructions executed?
- Tests for helper functions?
  - Which ones?

Generating new tests

- Fuzzing the uBPF runtime uncovered bugs
  - Permits comparing behavior of random programs between uBPF JIT, interpreter, and Linux Kernel runtime
- Manually reviewing the BPF ISA specification
  - Time consuming and error prone
- Generate from machine readable model of the BPF ISA specification

Open Questions

- Is this the best way to achieve this goal?
- Is it possible to check the compiler’s model of the BPF ISA (not just the verifier and runtime)?
- Ownership of this project
  - Currently within my personal GitHub
  - Approved to migrate to BPF Foundation
  - Some open legal questions remain