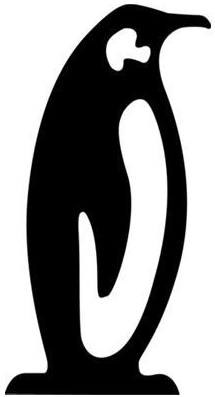


東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

Fuzzing the Verifier with a Test Oracle

東京 2025
LINUX
PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

BPF Test Oracle

1. Verifier False Negatives are Silent
2. State Embeddings
3. An In-Kernel BPF Test Oracle
4. Implementation
5. Example
6. Open Questions

Verifier False Negatives are Silent

Fuzzers are good at finding memory errors, deadlocks, kernel warnings, etc.

Struggle to find false negatives

- I.e., verifier accepts something it shouldn't
- By definition, typically silent

We need a test oracle to know when we reach unsound states

State Embeddings

OSDI'24 paper to turn make soundness bugs noisy

1. Start from accepted program

```
0: *(u64*)(r10 -40) = -1
1: r1 = *(u64*)(r10 -40)
2: r2 = 1
3: if r1 < 0 goto +1
4: r2 = 0
5: exit
```

State Embeddings

OSDI'24 paper to turn make soundness bugs noisy

1. Start from accepted program
2. Trace it to get concrete variable values

```
0: *(u64*)(r10 -40) = -1
1: r1 = *(u64*)(r10 -40)
2: r2 = 1
3: if r1 < 0 goto +1
4: r2 = 0
5: exit
```

State Embeddings

OSDI'24 paper to turn make soundness bugs noisy

1. Start from accepted program
2. Trace it to get concrete variable values
3. Fold concrete values and check them

```
0: r9 = 0
1: *(u64*)(r10 -40) = -1
2: r1 = *(u64*)(r10 -40)
3: r2 = 1
4: if r1 < 0 goto+1
5: r2 = 0
6: r9 += r1
7: r9 *= r2
8: if r9 != -1 goto+1
9: verifier_sink()
10: exit
```

State Embeddings

OSDI'24 paper to turn make soundness bugs noisy

1. Start from accepted program
2. Trace it to get concrete variable values
3. Fold concrete values and check them
4. Trigger verifier error if folded value is as expected
    a. By writing to R10

```
0: r9 = 0
1: *(u64*)(r10 -40) = -1
2: r1 = *(u64*)(r10 -40)
3: r2 = 1
4: if r1 < 0 goto+1
5: r2 = 0
6: r9 += r1
7: r9 *= r2
8: if r9 != -1 goto+1
9: verifier_sink()
10: exit
```

State Embeddings

- More complicated than it needs to be
  - Seems they tried to minimize kernel changes
- Limited precision due to folding
- Doesn't point directly to the issue

State Embeddings

- More complicated than it needs to be
  - Seems they tried to minimize kernel changes
- Limited precision due to folding
- Doesn't point directly to the issue
- Not open source
  - Used a whole new fuzzer
  - (Not clear that they fixed all the bugs they found)

An In-Kernel BPF Test Oracle

Idea from Dmitry Vyukov:

- Save states at verification, check at runtime against concrete values

Check directly the verifier's results, not some folded value

Bug reports can point directly to unexpected values

Implementation

- Limited scope:
  - Registers only: checks ranges & tnum
  - Runtime check only implemented in the interpreter

- Saves and test verifier data at pruning points

- Saves verifier data in array maps:
  - Read-only from userspace

- Rough first draft at https://lore.kernel.org/bpf/cover.1765158924.git.paul.chaignon@gmail.com

Example

```
0: r2 = r10;
1: *(u64*)(r2 -40) = -44;
2: r6 = *(u64*)(r2 - 40);
3: call 7;
4: r0 = 0;
5: exit;
```

```
  0: R1=ctx() R10=fp0
  0: (bf) r2 = r10                      ; R2=fp0 R10=fp0
  1: (7a) *(u64 *)(r2 -40) = -44        ; R2=fp0
                                          fp-40=0xffffffd4
  2: (79) r6 = *(u64 *)(r2 -40)         ; R2=fp0
                                          R6=0xffffffd4
                                          fp-40=0xffffffd4
  3: (85) call bpf_get_prandom_u32#7    ; R0=scalar()
  4: (b7) r0 = 0                        ; R0=0
  5: (95) exit
  processed 6 insns (limit 1000000) max_states_per_insn 0
total_states 0 peak_states 0 mark_read 0
```

Example

```
0: r2 = r10;
1: *(u64*)(r2 -40) = -44;
2: r6 = *(u64*)(r2 - 40);
3: call 7;
4: r0 = oracle_map[id:22]
5: r0 = 0;
6: exit;
```

Example

```
0: r2 = r10;
1: *(u64*)(r2 -40) = -44;
2: r6 = *(u64*)(r2 - 40);
3: call 7;
4: r0 = oracle_map[id:22]
5: r0 = 0;
6: exit;
```

```
# bpftool oracle dump id 22
State 0:
R0=scalar()
R6=scalar(u64=s64=u32=4294967252, s32=-44,
          var_off=(0xffffffd4; 0)

Found 1 state
```

Example

```
0: r2 = r10;
1: *(u64*)(r2 -4
2: r6 = *(u64*)(r
3: call 7;
4: r0 = oracle_ma
5: r0 = 0;
6: exit;
```

```
[   29.845786] ------------[ cut here ]------------
[   29.846696] oracle caught invalid states in
oracle_map[id:22]: r6=0xfffffffffffffd4
[   29.847857] WARNING: kernel/bpf/oracle.c:368 at 0x0,
CPU#1: minimal/544
[   29.849276] Modules linked in: xt_conntrack
xt_MASQUERADE xfrm_user ip6table_nat ip6table_filter
ip6_tables xt_set ip_set iptable_nat nf_nat xt_addrtype
iptable_filter overlay ip_tables
[   29.852689] CPU: 1 UID: 0 PID: 544 Comm: minimal Not
tainted 6.18.0-geb74985261f9 #1 PREEMPT(voluntary)
[   29.854589] Hardware name: QEMU Ubuntu 24.04 PC (i440FX
+ PIIX, 1996), BIOS 1.16.3-debian-1.16.3-2 04/01/2014
[   29.856501] RIP: 0010:oracle_test+0xf7/0x130
```

```
s32=-44,
```

Open Questions

- What instruction to hold the oracle maps?
  - Current LDIMM64 is clumsy

Example

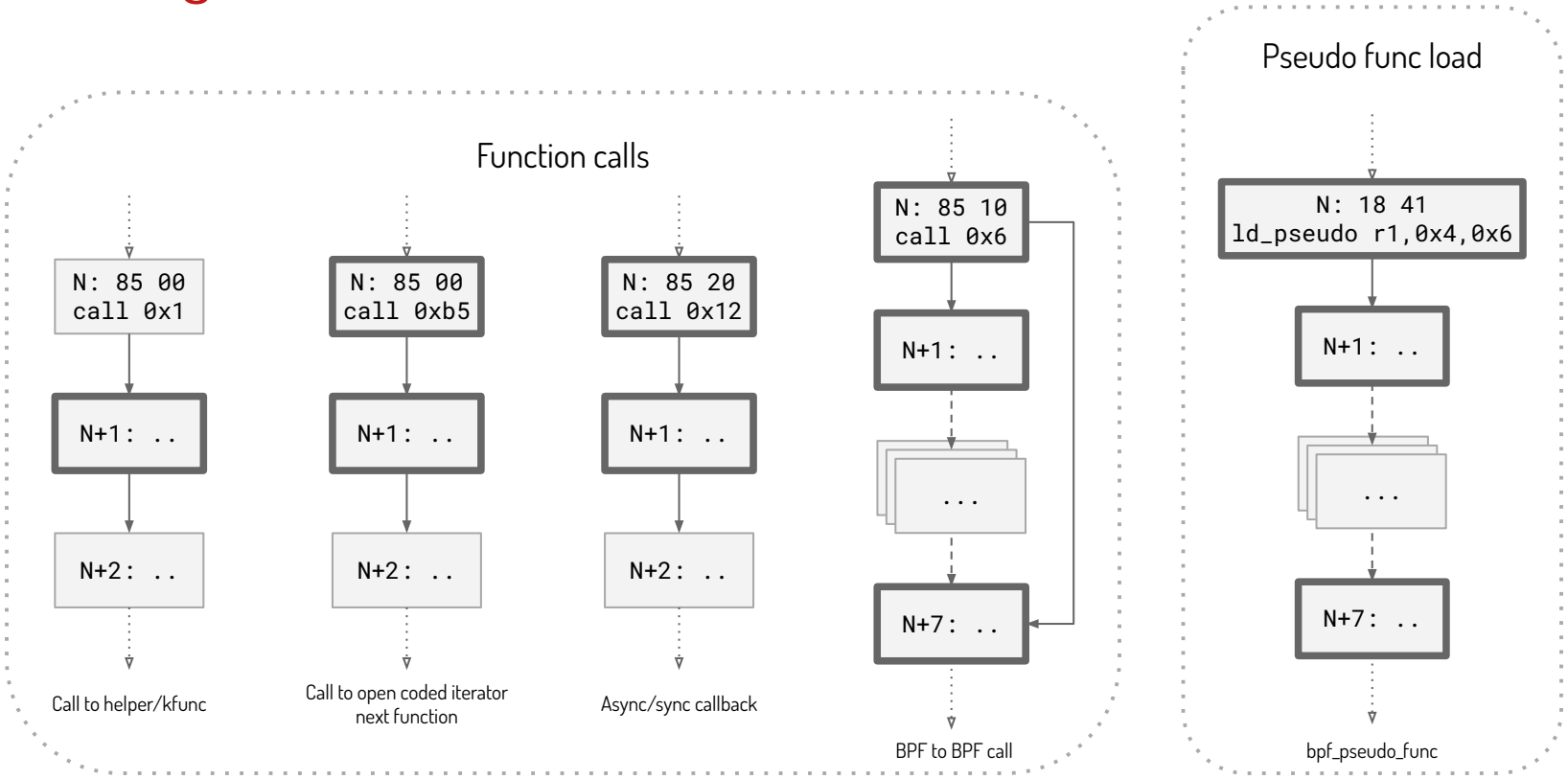
```
0: r2 = r10;
1: *(u64*)(r2 -40) = -44;
2: r6 = *(u64*)(r2 - 40);
3: call 7;
4: r0 = oracle_map[id:22]
5: r0 = 0;
6: exit;
```

Open Questions

- What instruction to hold the oracle maps?
  - Current LDIMM64 is clumsy

- When to run the test oracle?
  - Pruning points are convenient but maybe not optimal

Pruning Points



Open Questions

- What instruction to hold the oracle maps?
  - Current LDIMM64 is clumsy
- When to run the test oracle?
  - Pruning points are convenient but maybe not optimal
- Subprogs not supported yet
  - Without callchain at runtime, we lose some precision

Open Questions

- What instruction to hold the oracle maps?
  - Current LDIMM64 is clumsy
- When to run the test oracle?
  - Pruning points are convenient but maybe not optimal
- Subprogs not supported yet
  - Without callchain at runtime, we lose some precision
- What else can we test? Pointers?

Open Questions

```
struct bpf_reg_oracle_state {
    bool scalar;
    bool ptr_not_null;

    struct tnum var_off;
    s64 smin_value;
    s64 smax_value;
    u64 umin_value;
    u64 umax_value;
    s32 s32_min_value;
    s32 s32_max_value;
    u32 u32_min_value;
    u32 u32_max_value;
};
```

Open Questions

- What instruction to hold the oracle maps?
  - Current LDIMM64 is clumsy
- When to run the test oracle?
  - Pruning points are convenient but maybe not optimal
- Subprogs not supported yet
  - Without callchain at runtime, we lose some precision
- What else can we test? Pointers?

Conclusion

- Test oracle to expose false negatives to fuzzers
- RFC at https://lore.kernel.org/bpf/cover.1765158924.git.paul.chaignon@gmail.com