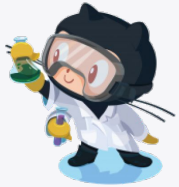
The State of eBPF Fuzzing

Paul Chaignon | @pchaigno

Software Engineer, Isovalent at Cisco

ISOVALENT

Who Am I?



Paul Chaignon
Software Engineer @ Isovalent / Cisco
Datapath team for Cilium

Interested in BPF fuzzing for a while:

- First contributions to verifier after rebasing bpf-fuzzer
- Updating BPF descriptions in Syzkaller since 2019

eBPF Fuzzing

- Syzkaller coverage
- Recent improvements
- Other approaches
- Finding a test oracle
- Conclusion

ISOVALENT

Syzkaller

- Well maintained and well integrated
  - Lots of up-to-date syscall descriptions
  - Continuously running and reporting bugs (syzbot)
- Code coverage guided fuzzer (kcov)
- Structured fuzzer: syzlang descriptions of syscalls
- Can find many bugs via kernel sanitizers (KASAN & co.)
- Runs full kernels in VMs

ISOVALENT

Syzkaller's Coverage

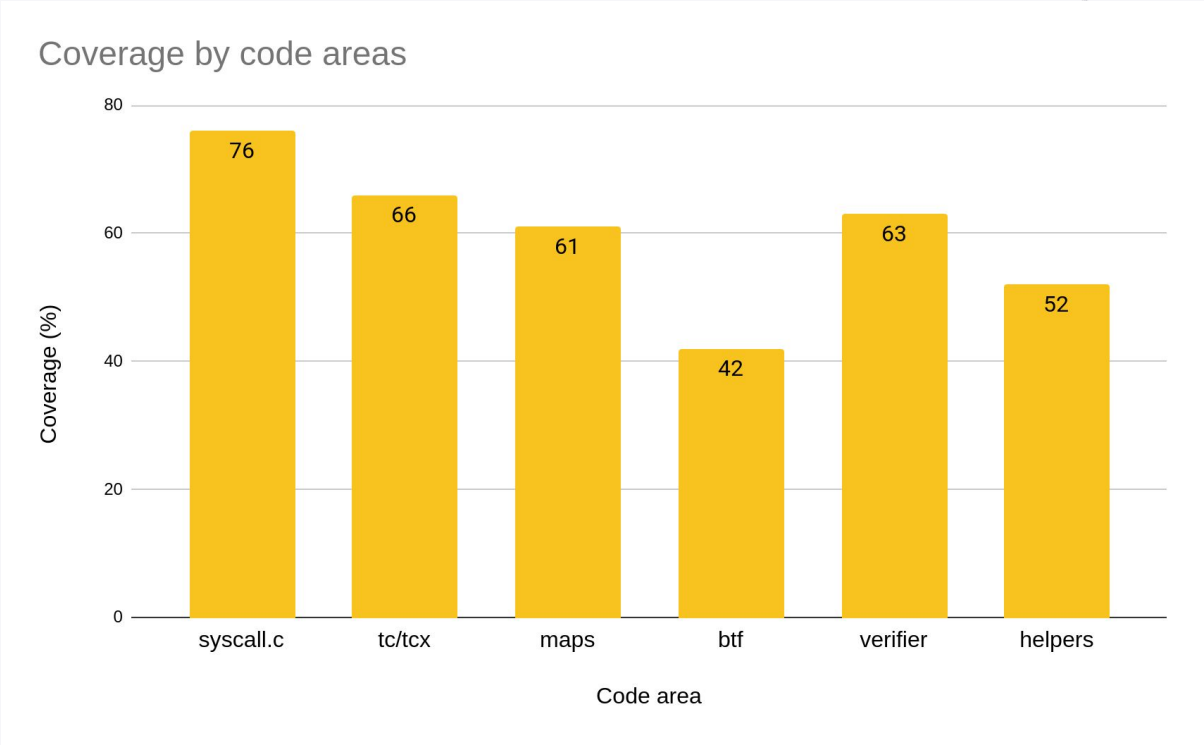
- Syzkaller finds many bugs in BPF
- But doesn't tell much on effectiveness
- Code coverage is a better measure

WARNING in vmap_pages_range_noflush (2)	bpf net
BUG: unable to handle kernel paging request in bpf_prog_A...	
WARNING in sock_map_close (2)	bpf net
possible deadlock in lock_timer_base	net bpf
KASAN: slab-use-after-free Read in htab_map_alloc (2)	bpf
KASAN: stack-out-of-bounds Read in xdp_do_check_flushed...	
general protection fault in __dev_flush	net bpf
general protection fault in bq_flush_to_queue	bpf net
KASAN: slab-use-after-free Read in bq_xmit_all	bpf net
general protection fault in __xsk_map_flush	bpf net
general protection fault in __cpu_map_flush	bpf net
general protection fault in xdp_do_generic_redirect	net bpf
general protection fault in dev_map_redirect	bpf net
stack segment fault in dev_hash_map_redirect	bpf net
general protection fault in xdp_do_redirect	bpf net
stack segment fault in bpf_xdp_redirect	bpf net
stack segment fault in cpu_map_redirect	net bpf
WARNING in bpf_lwt_seg6_adjust_srh	bpf net
possible deadlock in console_flush_all (2)	trace bpf
WARNING in skb_ensure_writable	bpf net
INFO: task hung in bpf_prog_dev_bound_destroy	bpf
possible deadlock in __sock_map_delete	bpf net
general protection fault in bpf_get_attach_cookie_tracing ...	
KASAN: slab-use-after-free Read in bpf_link_free (2)	bpf
WARNING in sock_map_close	bpf net
possible deadlock in sock_hash_delete_elem (2)	bpf net

ISOVALENT

Syzkaller's Coverage

- From syzbot
- Rough aggregation

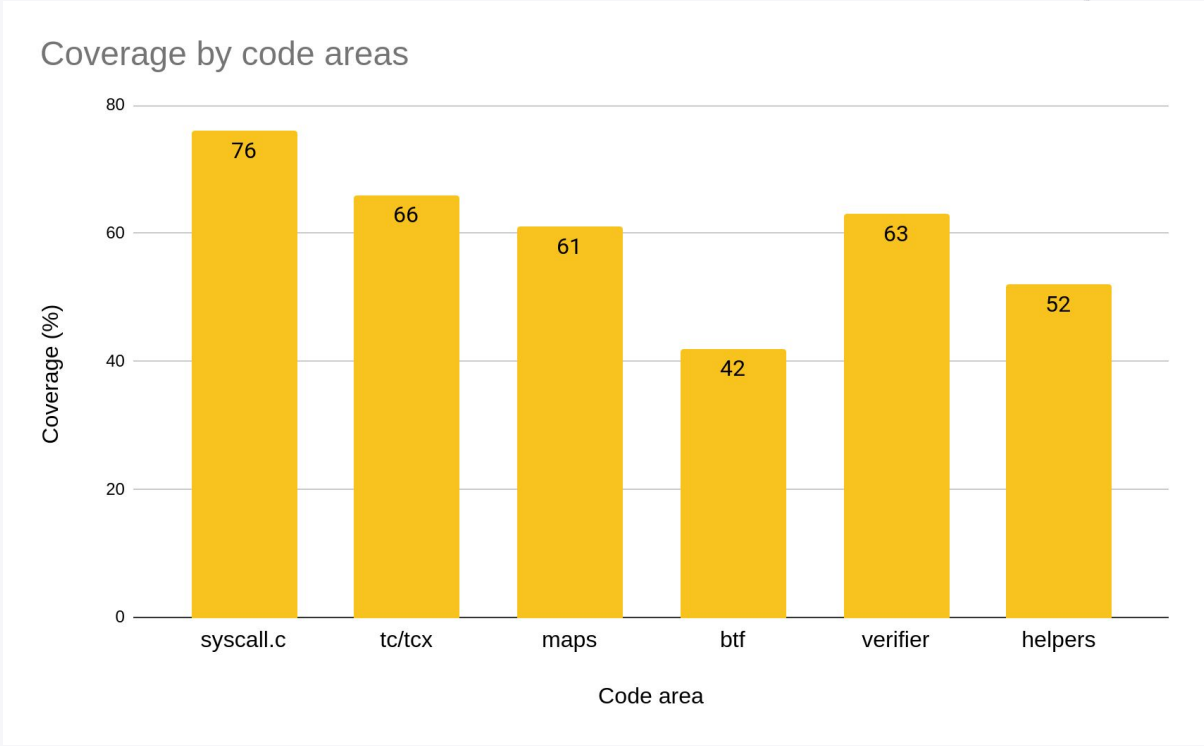


ISOVALENT

Syzkaller's Coverage

- From syzbot
- Rough aggregation

- Closer to syscall is better
- BTF desc. is outdated
- Helpers are hard to reach



Challenges of Fuzzing BPF

- Many dependencies between various part of the input
  - Ex. program type and allowed helpers
  - Ex. sizes between map creation and map value load
  - Ex. jump offset and program structure
  - Ex. ordering between write and read of R0
  - Ex. BTF kfunc prototypes and kfunc calls

- Hard to describe with a simple description language
  - Don’t want to reimplement the verifier in the fuzzer 😵

- Several layers to pass:
  - Ex. kfunc call requires valid BTF + valid program + valid exec syscall

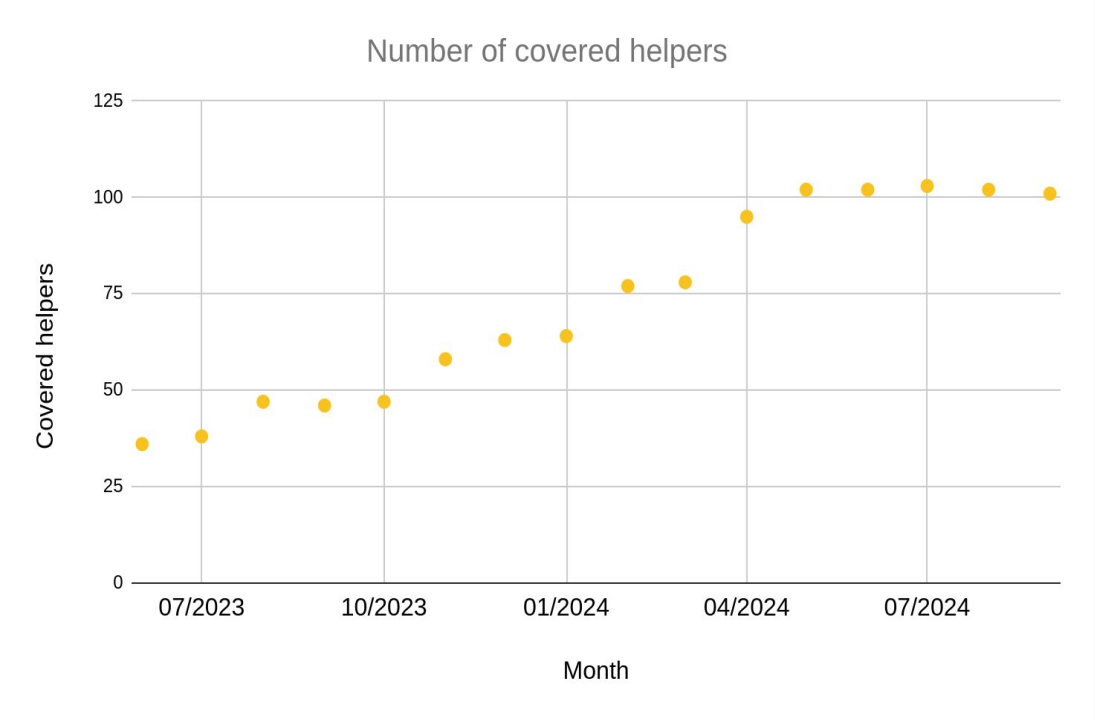
eBPF Fuzzing

- Syzkaller coverage
- **Recent improvements**
- Other approaches
- Finding a test oracle
- Conclusion

ISOVALENT

Recent Improvements

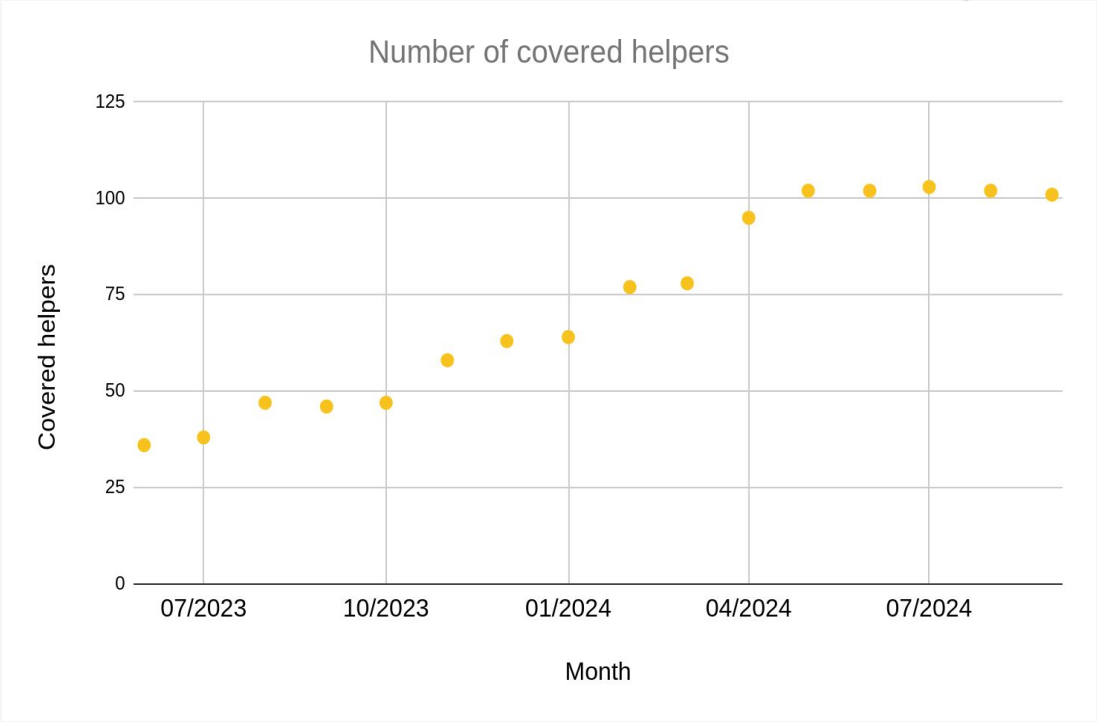
- Counting all covered helpers, even partially covered
- Covered helpers doubled in a year (+55)



ISOVALENT

Recent Improvements

- Counting all covered helpers, even partially covered
- Covered helpers doubled in a year (+55)
- Describing full helper calls paid off
- Described only 8 helpers, syzkaller guessed the rest



ISOVALENT

Recent Improvements

- Syzlang becoming more expressive with conditional fields
- Enabled more precision in BPF descriptions

```
link_create_netkit {
        relative_link_fd        fd_bpf_link     (if[value[bpf_link_create_arg_t:flags] & BPF_F_LINK_OR_ID == BPF_F_LINK])
        relative_prog_fd        fd_bpf_prog     (if[value[bpf_link_create_arg_t:flags] & BPF_F_LINK_OR_ID == 0])
        relative_link_id        bpf_link_id     (if[value[bpf_link_create_arg_t:flags] & BPF_F_LINK_OR_ID == BPF_F_LINK_OR_ID])
        relative_prog_id        bpf_prog_id     (if[value[bpf_link_create_arg_t:flags] & BPF_F_LINK_OR_ID == BPF_F_ID])
        exp_revision            bpf_revision
} [packed]
```

eBPF Fuzzing

- Syzkaller coverage
- Recent improvements
- **Other approaches**
- Finding a test oracle
- Conclusion

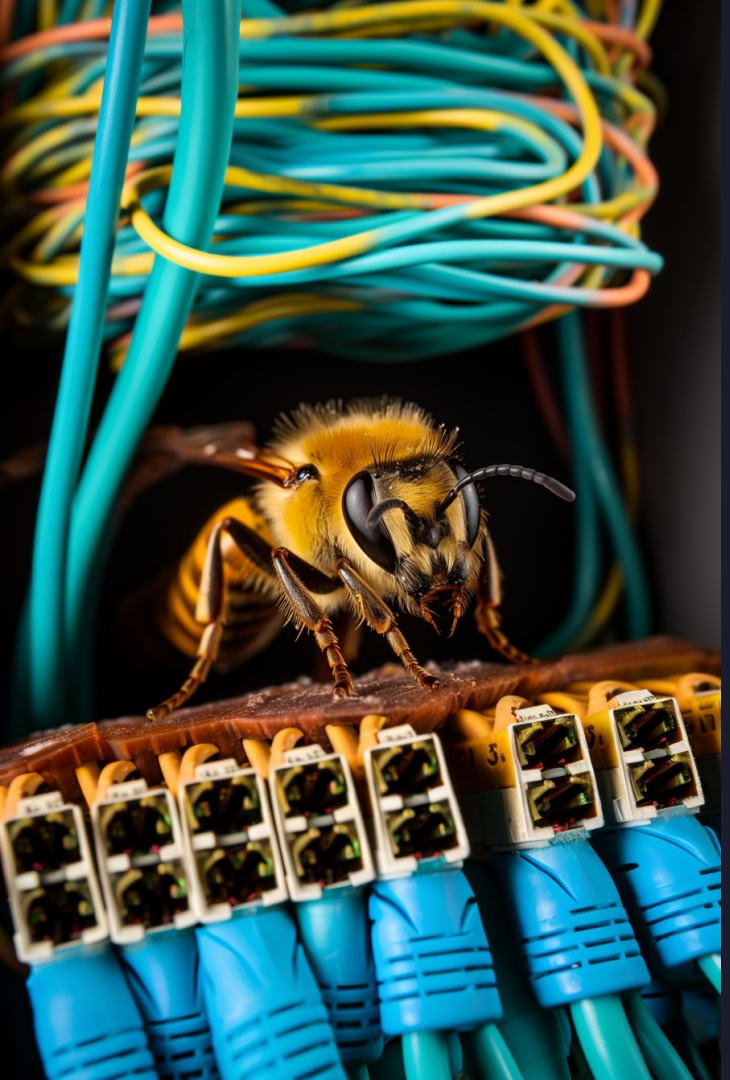
Running the Verifier in Userspace

- bpf-fuzzer by Facebook: first-ever eBPF fuzzer
  - Uses libfuzzer
  - Verifier executed in userspace with lots of glue code
- kBdysch by Anatoly Trosinenko
  - Uses AFL
  - Relies on LKL instead of manual port to userspace
- High maintenance cost!
  - Did find multiple bugs though

ISOVALENT

Buzzer: Tailored Fuzzing for BPF

- Runs the kernel in VMs like syzkaller
- Somewhat focus on the verifier
- BPF-specific fuzzing strategies
  - Attempting out-of-bound map writes
  - Checking verifier logs
  - Or plain old coverage-based
- Found two vulnerabilities so far
- Focus of the next talk!



eBPF Fuzzing

- Syzkaller coverage
- Recent improvements
- Other approaches
- **Finding a test oracle**
- Conclusion

ISOVALENT

We're Missing a Test Oracle!

- Good at finding memory errors, crashes, deadlocks, kernel warnings, etc.
- Struggle to find verifier bypasses
  - Because verifier bypasses are typically silent
- Need a test oracle for the verifier's soundness

State Embeddings as a Test Oracle

- Hao Sun and Zhendong Su devised one test oracle for the verifier
- Turn silent soundness issues into loud verifier errors
- Published at OSDI'24

ISOVALENT

State Embeddings as a Test Oracle

1. Start from accepted BPF 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
```

ISOVALENT

State Embeddings as a Test Oracle

1. Start from accepted BPF program
2. Fold variables into single register

```
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
```

ISOVALENT

State Embeddings as a Test Oracle

1. Start from accepted BPF program
2. Fold variables into single register
3. Trigger verifier error if folded value is as expected
   a. Ex. write 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
```

ISOVALENT

State Embeddings as a Test Oracle

1. Start from accepted BPF program
2. Fold variables into single register
3. Trigger verifier error if folded value is as expected
   a. Ex. write to R10
4. Modified program passes verifier implies:
   a. concrete folded value ∉ abstract folded value
   b. Ex. -1 ∉ verifier's view of R9
   c. That is, unsoundness issue!

```
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
```

eBPF Fuzzing

- Syzkaller coverage
- Recent improvements
- Other approaches
- Finding a test oracle
- **Conclusion**

ISOVALENT

Conclusion

- Help welcome for syzkaller descriptions!
  - Lots to do, easy and harder stuff
  - Very helpful & responsive maintainers
  - Bugs as rewards
- IMO, we should converge approaches in syzkaller
  - Better integration with the kernel (syzbot)
- How can we improve the status quo?
  - Can we use state embeddings in syzkaller? In buzzer?

ISOVALENT

Thanks !