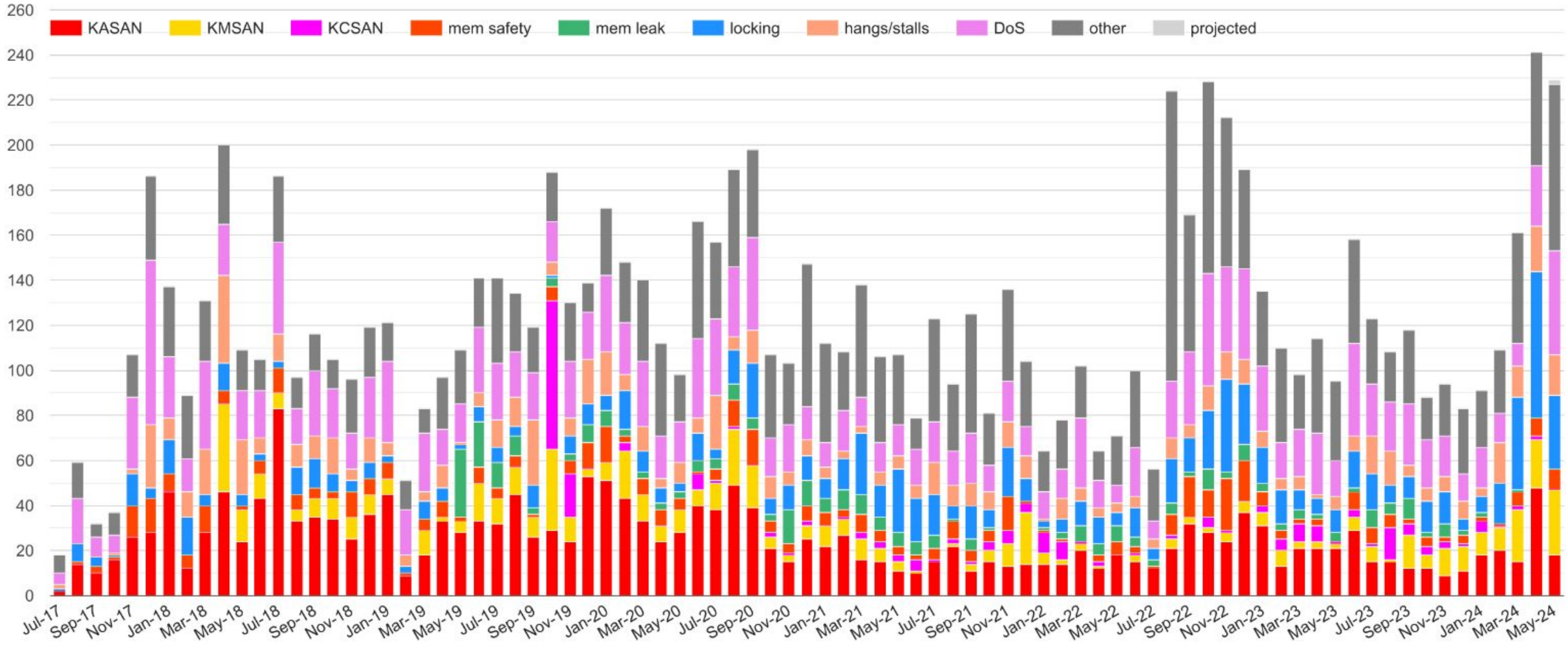
# SymBisect: Accurate Bisection for Fuzzer-Exposed Vulnerabilities

Zheng Zhang[1], Yu Hao[1], Weiteng Chen[3], Xiaochen Zou[1], Xingyu Li[1], Haonan Li[1], Yizhuo Zhai[1], Zhiyun Qian[1], Billy Lau[2]

[1]UC Riverside, [2]Google Inc, [3]Microsoft Research

# Thousands of bugs reported with fuzzers

> 6000 bugs reported to Linux kernel Mainline by Syzbot*




* Syzbot: 7 years of continuous kernel fuzzing- Aleksandr Nogikh, Google, 2023
Figure by Dmitry Vyukov

# Bug Bisection

- Automating analysis of fuzzer-exposed bugs.
- Bisection: identifying the commit that introduces the bug.

- Benefits:
  1. Accelerate bug analysis and patch development.

     With bisection: 39% addressed in 45 days.

     Without bisection: 19% addressed in 45 days.[1]
  2. Identify vulnerable software versions.

     Inform users about whether they need to worry about updating their software. [2]

[1] Syzbot: 7 years of continuous kernel fuzzing
[2] V-szz: automatic identification of version ranges affected by CVE vulnerabilities

# Bisection with PoC: Limitations

- Many kernel versions do not build/boot with syzbot config.
- Bug reproducers (PoC) are not always reliable.
- Single reproducer might trigger unrelated bugs.
- Only 50% accuracy in a previous study conducted by the syzbot team.

# Bisection with Patch: Limitations

- Require the patch.
- Bug-inducing commit may not change the patch functions.
- Rely on heuristics which are inherently imprecise.
  - e.g., deleted line in the patch exists in target version => vulnerable

# Motivating example

The Bug-inducing Commit:

```
static struct bpf_map *htab_map_alloc(...)
1  - cost = S1*C1 + S2*S3;
     ......
2  - cost += S2*C2
3  - err = bpf_map_charge_init(..., cost);
4  - if (err)
   -     goto free_htab;
     ......
5    err = prealloc_init(...);

int bpf_map_charge_init(...,u64 size)
     ......
6    if (size >= U32_MAX - PAGE_SIZE)
        return -E2BIG;
```

The Patch:

```
static int prealloc_init(...)
     S3 = S3 + C2;
     ......
7  - htab->elems =bpf_map_area_alloc(S2*S3,
8  + htab->elems =bpf_map_area_alloc((u64)S2*S3,
```

```
S1: (u64)htab->n_buckets      C1: sizeof(struct bucket)
S2: (u64)htab->elem_size      C2: num_possible_cpus()
S3: htab->map.max_entries
```

**Symbolic execution trace (partly):**

**...... -> htab_map_alloc() -> bpf_map_charge_init() -> prealloc_init() -> ......**

Before inducing commit:

```
Line1  Assignment: cost = S1*C1 + S2*S3
Line2  Assignment: cost +=  S2*C2
Line6  Constraint  S1*C1 + S2(S3+C2) < U32_Max - 4096
Line7  Overflow condition: S2(S3+C2) > U32_Max
Not solvable => Not vulnerable
```

After inducing commit (before patch):

```
Line8  Overflow condition: S2(S3+C2) > U32_Max
Solvable => Vulnerable
```

- Bisection with PoC: trigger an unrelated bug.
- Bisection with Patch: bug-inducing commit does not alter the patch function.
- Symbolic execution: succeed.

# Challenge: Scalability

- Path explosion

  The number of forked states may grow exponentially as the execution progresses.

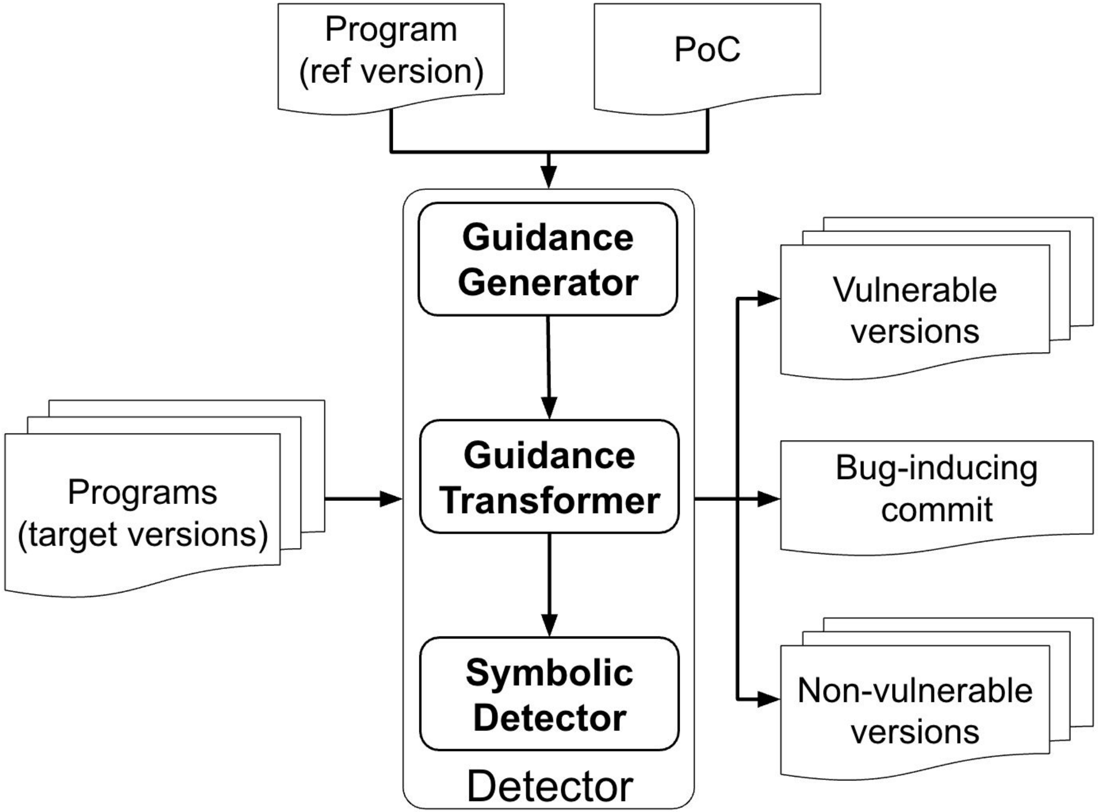
  Especially serious in complex programs such as Linux kernel.

- Key Observations

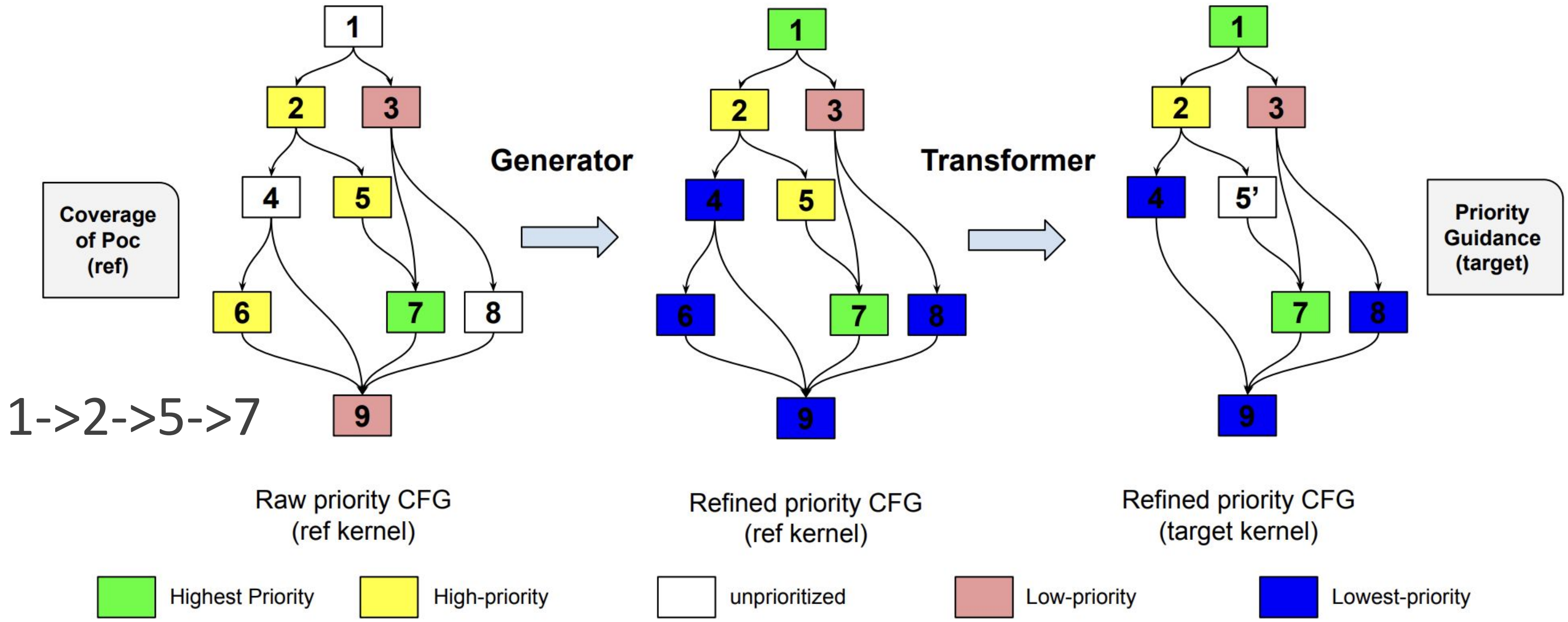
  Leverage fine-grained trace-level information about how the vulnerability is triggered.

  Such coverage information can help prioritize relevant execution paths.

# System Architecture



# Guidance Generation




- Call Stack Guidance: highest/lowest priority
- Path Guidance: high/low priority

# Implementation

- Guidance Generator/Transformer.

  4726 LoC Python.

- Symbolic Detector. (Based on KLEE)

  Modifications to KLEE to better handle symbolic variables (symbolic addresses, symbolic sizes, etc.)

  4347 LoC C++.

# Evaluation: Accuracy

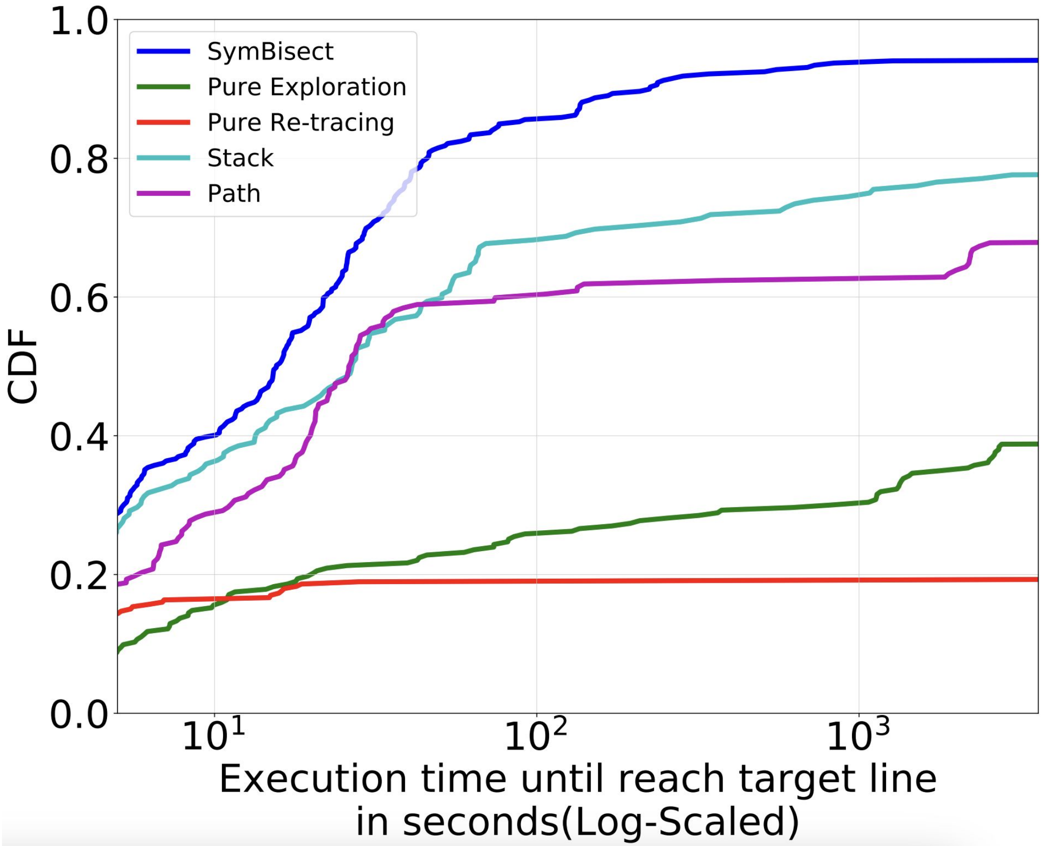
| Tools | TP | FP | TN | FN | Accuracy | Precision | Recall | F-1 Score |
|---|---|---|---|---|---|---|---|---|
| SymBisect | 237 | 29 | 348 | 31 | 90.7% | 89.1% | 88.4% | 88.7% |
| Syzbot(PoC) | 146 | 27 | 350 | 122 | 76.9% | 84.4% | 54.5% | 66.2% |
| V0Finder | 138 | 0 | 377 | 130 | 79.8% | 100.0% | 51.5% | 68.0% |
| VSZZ | 250 | 145 | 232 | 18 | 74.7% | 63.4% | 93.3% | 75.4% |

Table 1: **The results of vulnerable versions detection**

| Tools | Correct | Incorrect | Accuracy |
|---|---|---|---|
| SymBisect | 24 | 8 | 75% |
| Syzbot | 16 | 16 | 50% |
| V0Finder | 11 | 21 | 34.375% |
| VSZZ | 18 | 14 | 56.25% |

Table 2: **The results of bug-inducing commit identification**

# Evaluation: Performance



# Q&A

Thanks for your attention!