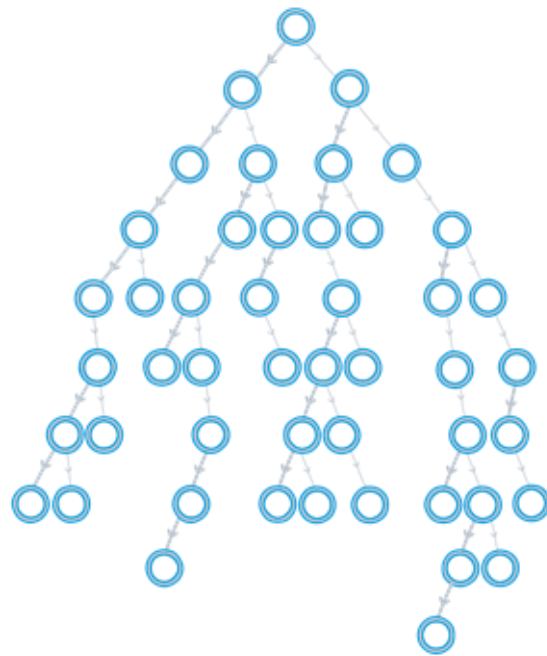


Generation and Validation of Summaries for File Systems

A KLEE experiment



Frederico Ramos · João Matos · Jorge Hristovsky · Pedro Adão · Rúben Martins · José Fragoso Santos

Symbolic Execution

KEE



TRILON

Gillan

S²E

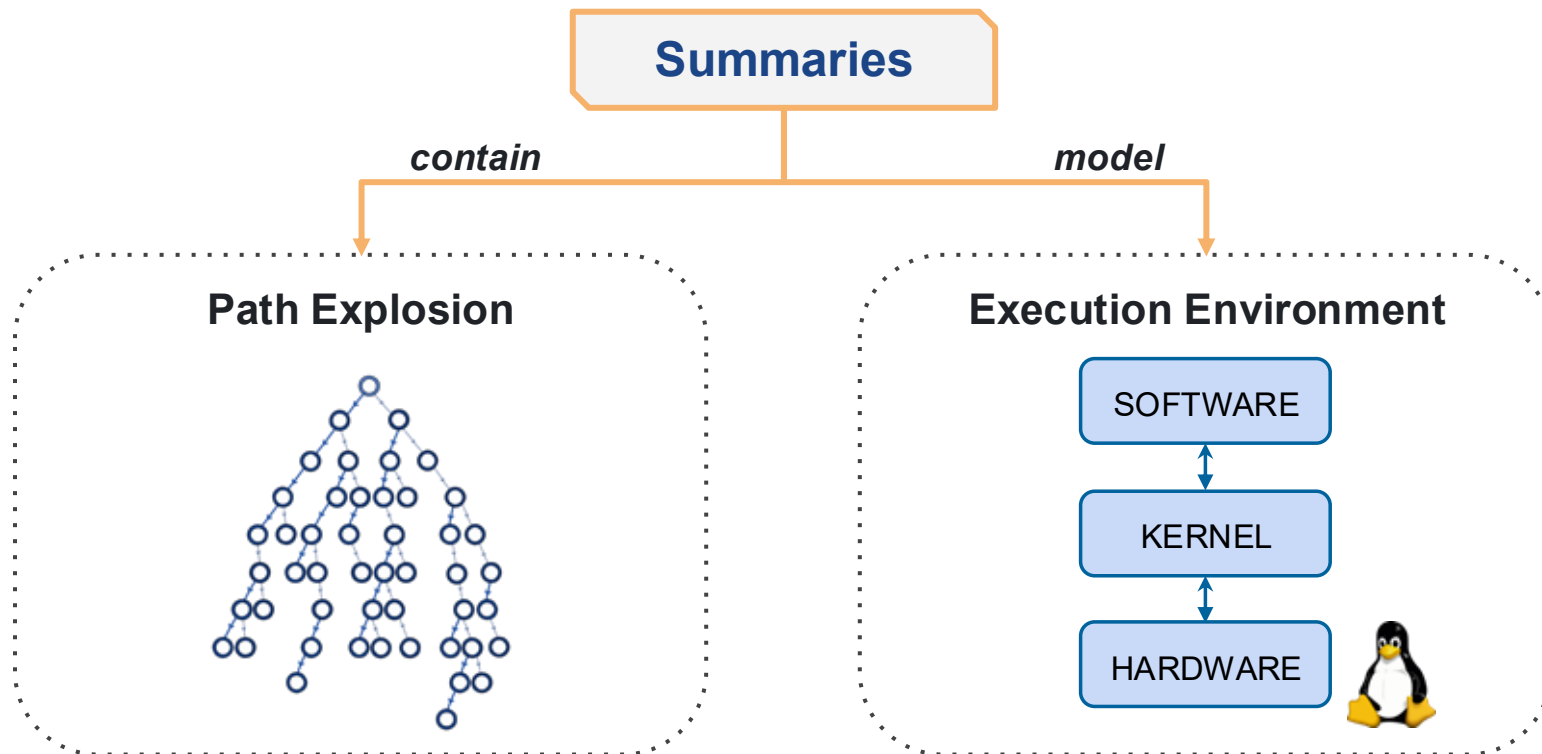


DIVINE 4

Cloud9

 BINSEC

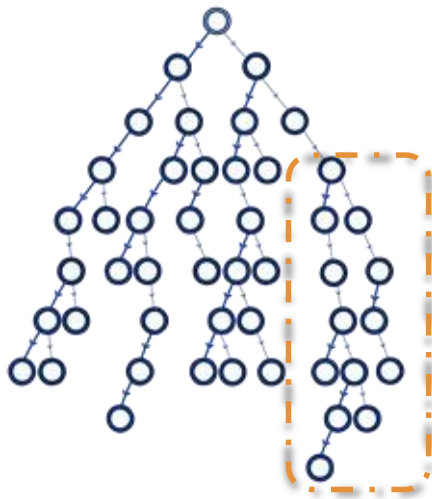
Symbolic Execution



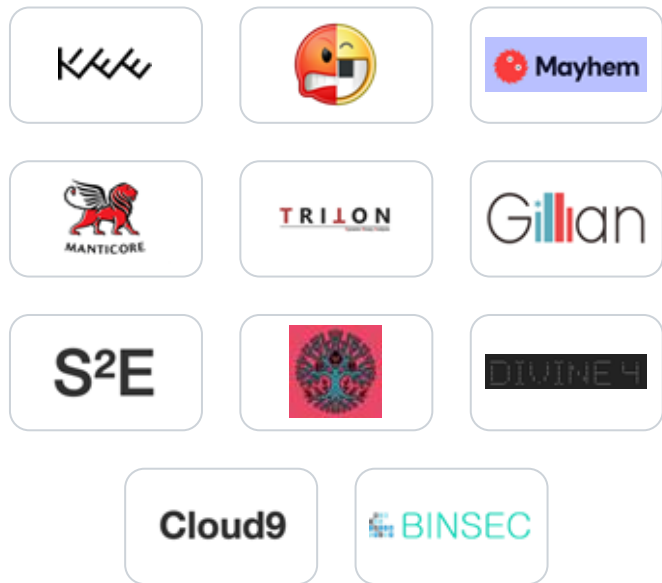
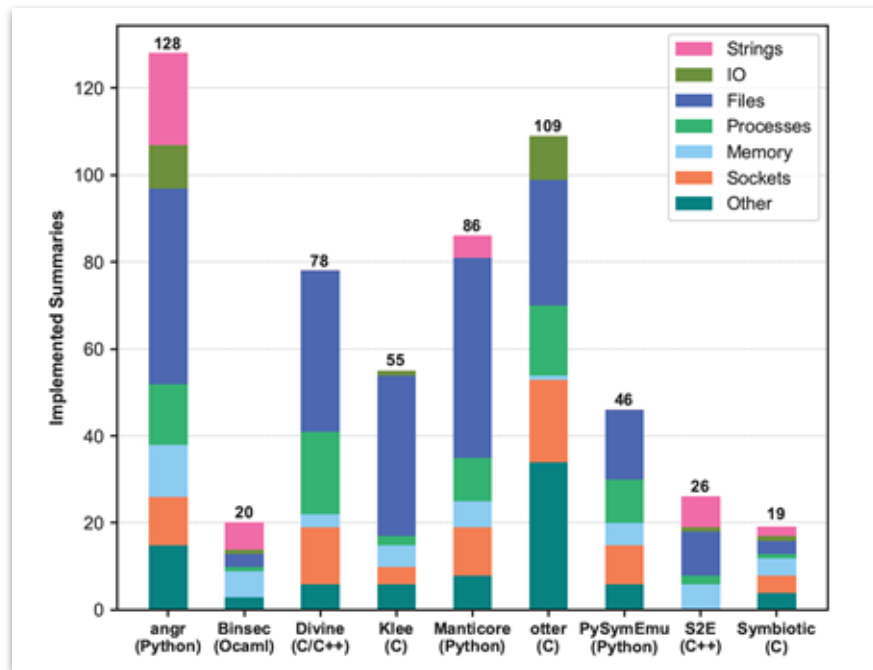
Symbolic Summaries

Simulate

function behaviour by
manipulating symbolic
states

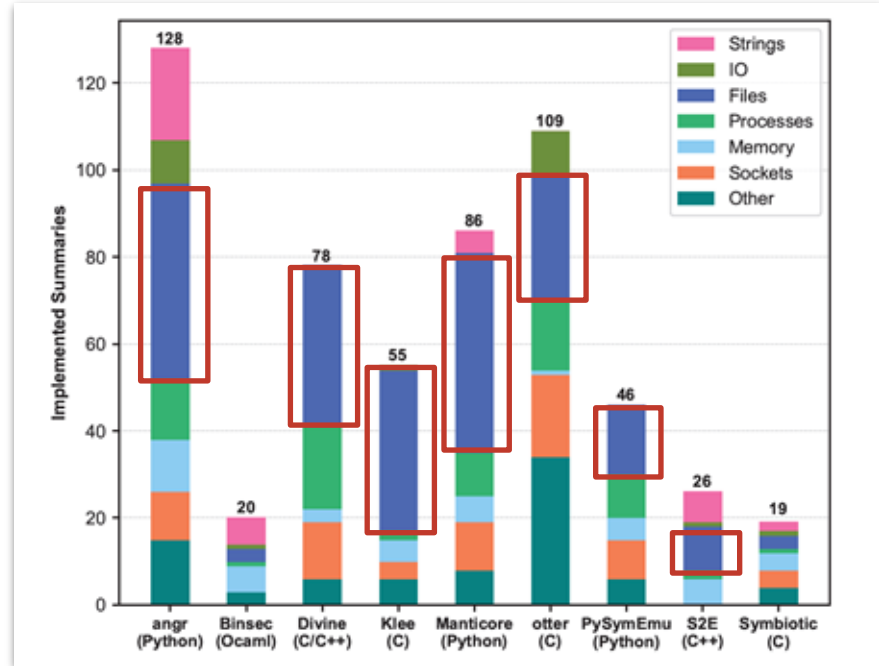


Support for Files in SE Tools



Support for Files in SE Tools

Tools seek to model **file** functions

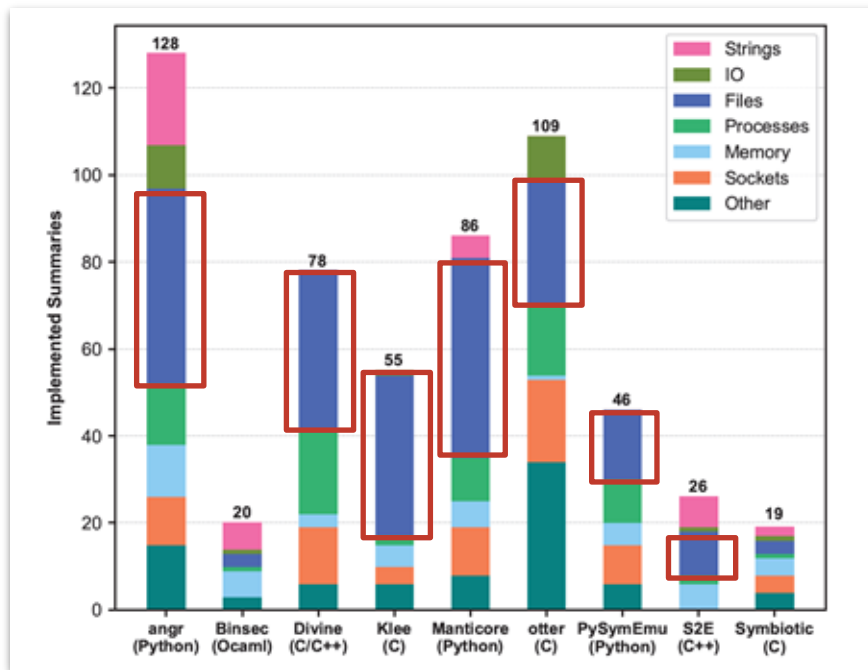


Support for Files in SE Tools

Tools seek to model **file** functions



Problem: hard to implement and validate



Support for Files in SE Tools

Current Gap

- 1 Error-prone and complex to design
- 2 The existing summary validation tool does **not** support files

Support for Files in SE Tools

Current Gap

- 1 Error-prone and complex to design
- 2 The existing summary validation tool does **not** support files

Bugs can be present in both the summary code and the underlying file system model

write summary in KLEE

```
328 ssize_t read(int fd, void *buf, size_t count) {
329     static int n_calls = 0;
330     exe_file_t *f;
331
332     n_calls++;
333
334     if (count == 0)
335         return 0;
336
337     if (buf == NULL) {
338         errno = EFAULT;
339         return -1;
340     }
341
342     f = __get_file(fd);
343
344     if (!f) {
345         errno = EBADF;
346         return -1;
347     }
348
349     if (__exe_fs.max_failures && n_calls == n_calls) {
350         __exe_fs.max_failures--;
351         errno = EIO;
352         return -1;
353     }
354
355     if (!f->isfile) {
356         /* concrete file */
357         int r;
358         buf = __concretize_ptr(buf);
359         count = __concretize_size(count);
360         /* XXX In terms of looking for bugs we really should do this check
361          * before concretization, at least once the routine has been fixed
362          * to properly work with symbolics. */
363         klee_check_memory_access(buf, count);
364         if (f->fd == 0)
365             r = syscall(__NR_read, f->fd, buf, count);
366         else
367             r = syscall(__NR_pread64, f->fd, buf, count, (off64_t) f->off);
368
369         if (r == -1)
370             return -1;
371
372         if (f->fd != 0)
373             f->off += r;
374         return r;
375     }
376
377     else {
378         if (!f->flags & O_RDONLY) {
379             errno = EBADF;
380             return -1;
381         }
382
383         assert(f->off >= 0);
384         if (((off64_t)f->off->size) < f->off)
385             return 0;
386
387         /* symbolic file */
388         if (f->off + count > f->off->size) {
389             count = f->off->size - f->off;
390         }
391
392         memcpy(buf, f->off->contents + f->off, count);
393         f->off += count;
394
395         return count;
396     }
397 }
```

Support for Files in SE Tools

Previous Work

SumBoundVerify

Bounded validation of symbolic summaries

- Tool-independent summaries via a **symbolic reflection API**
- Validated 37 summaries of angr, Binsec and Manticore: **24 buggy**

Toward Tool-Independent Summaries for Symbolic Execution

Frederico Ramos ☒☑

Instituto Superior Técnico, University of Lisbon, Portugal
INESC-ID Lisbon, Portugal

Nuno Sabino ☒☑

Instituto Superior Técnico, University of Lisbon, Portugal
Carnegie Mellon University, Pittsburgh, PA, USA
Institute of Telecommunications, Campus de Santiago, Aveiro, Portugal

Pedro Adão ☒☑

Instituto Superior Técnico, University of Lisbon, Portugal
Institute of Telecommunications, Campus de Santiago, Aveiro, Portugal

David A. Naumann ☒☑

Stevens Institute of Technology, Hoboken, NJ, USA

José Fragoso Santos ☒☑

Instituto Superior Técnico, University of Lisbon, Portugal
INESC-ID Lisbon, Portugal

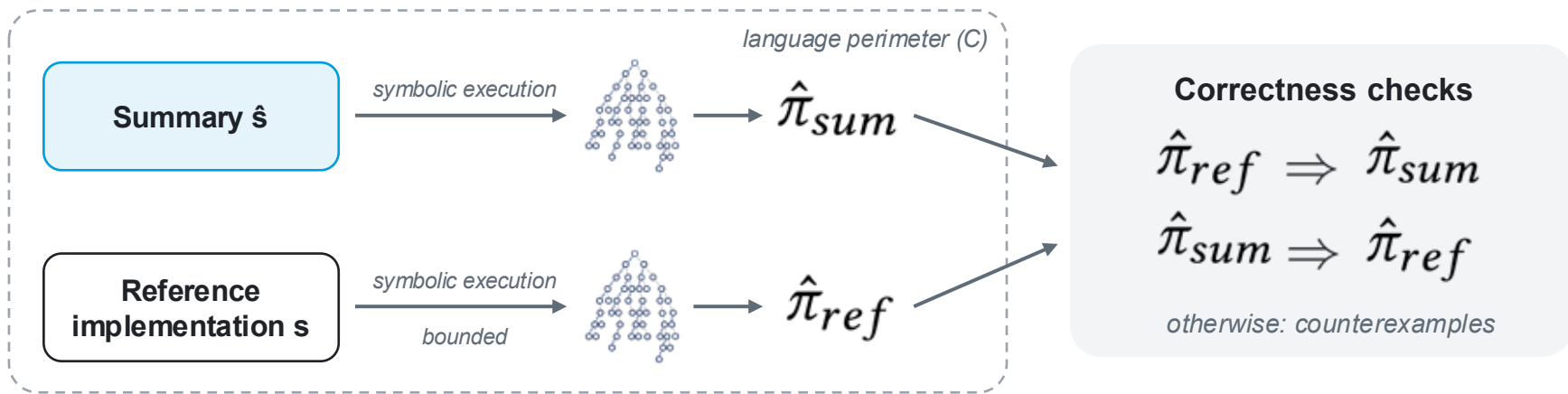
Abstract

We introduce a new symbolic reflection API for implementing tool-independent summaries for the symbolic execution of C programs. We formalise the proposed API as a symbolic semantics and extend two state-of-the-art symbolic execution tools with support for it. Using the proposed API, we implement 67 tool-independent symbolic summaries for a total of 26 libc functions. Furthermore, we present SUMBOUNDVERIFY, a fully automatic summary validation tool for checking the bounded correctness of the symbolic summaries written using our symbolic reflection API. We use SUMBOUNDVERIFY to validate 37 symbolic summaries taken from 3 state-of-the-art symbolic execution tools, *angr*, *Binsec* and *Manticore*, detecting a total of 24 buggy summaries.

Support for Files in SE Tools

Previous Work: How *SumBoundVerify* works

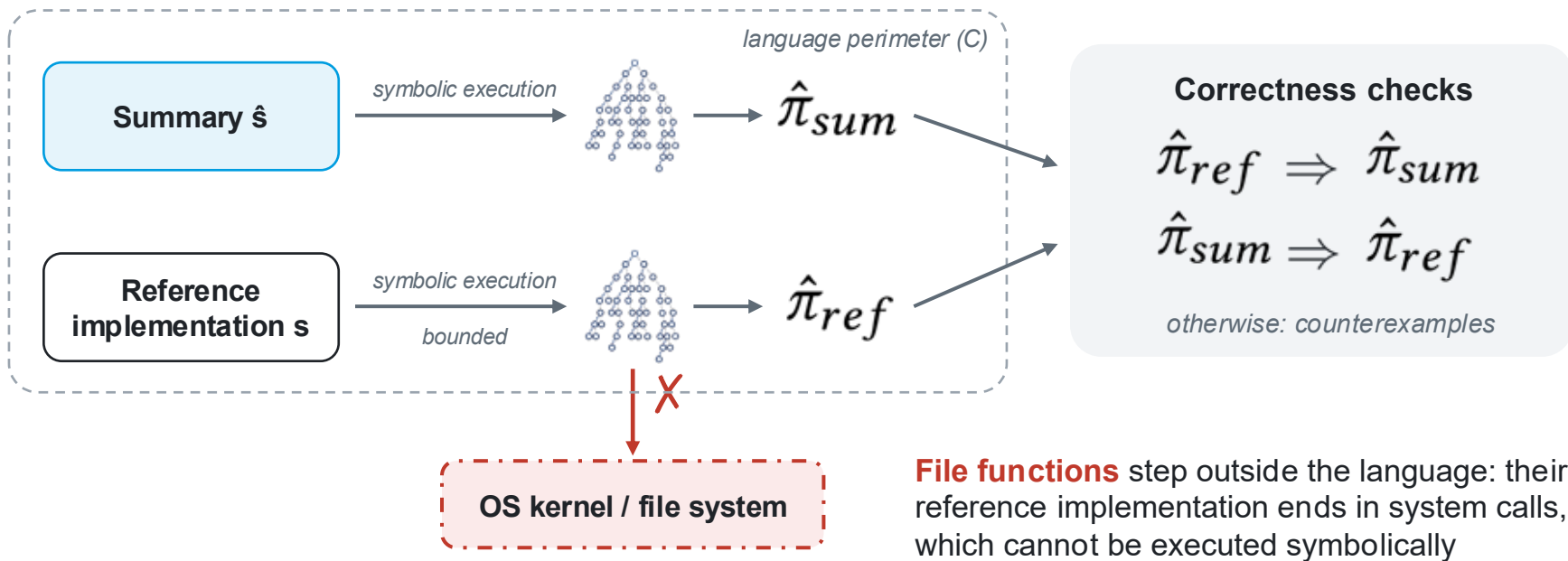
Compares the paths of the **summary** with the paths of a **reference implementation**



Support for Files in SE Tools

Previous Work: How *SumBoundVerify* works

Compares the paths of the **summary** with the paths of a **reference implementation**



Contributions

- 1 A symbolic reflection API with support for file-system functions**
 - Implemented in *angr* and *KLEE*
- 2 A summary validation tool with support for file summaries**
 - libc file test suite
 - Fuzzing engine
- 3 An agentic tool for summary generation**
 - Off-the-shelf coding agents
 - Templates + validation feedback

Symbolic Reflection API

Extends the SumBoundVerify API with primitives for file-system summaries

Group I *new*

File-system operations

```
__file_open  
__file_create  
__file_close  
__file_read / __file_write  
__file_dup / __file_dup2  
__file_set_size, ...
```

Group II *new*

Reflection & inspection

```
__file_size  
__file_offset  
__file_flags  
__file_mode  
__file_exists  
__get_errno, ...
```

Group III *from SumBoundVerify*

Symbolic reasoning

```
__sym_var  
__assume  
__assert  
__is_sat / __is_certain  
__EQ, __LT, __AND_, ...  
__concretize
```

39 primitives

implemented on top of:



angr

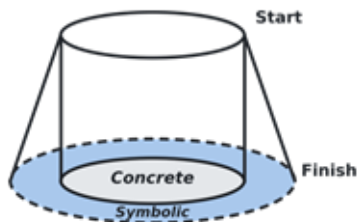


KLEE

Validation

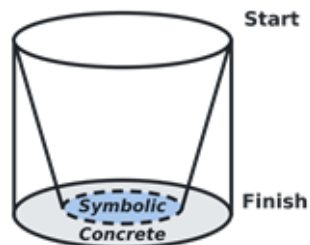
Summary correctness properties

Over-approximating



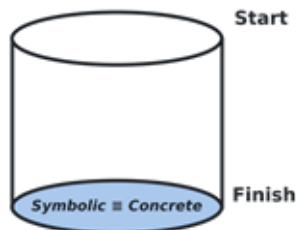
Models **all** concrete paths,
possibly more

Under-approximating



Models **only** concrete
paths, possibly fewer

Exact



Models **all and only** the
concrete paths

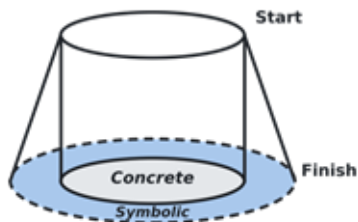
Target: over-approximation

Essential for security analyses that aim to guarantee the absence of vulnerabilities

Validation

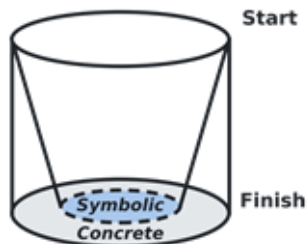
Summary correctness properties

Over-approximating



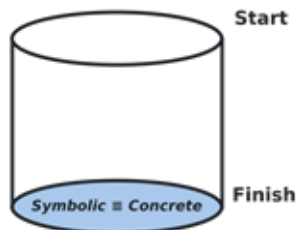
Models **all** concrete paths,
possibly more

Under-approximating



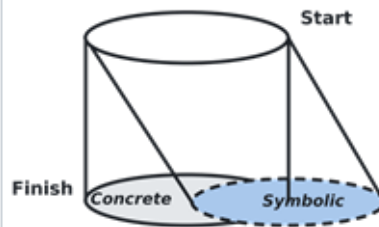
Models **only** concrete
paths, possibly fewer

Exact



Models **all and only** the
concrete paths

Buggy



Misses concrete paths **and**
models spurious ones

Bug: A summary that is neither under- nor over-approximating

Validation

Symbolic test suite

127 symbolic tests targeting
libc file functions

open / close 61 tests

read / write 25 tests

lseek 15 tests

chmod 13 tests

dup / dup2 13 tests

Tests are written with the **API primitives**, so they run on any engine that implements the API

```
// Write 5 bytes with O_WRONLY should succeed
int main(void) {
    char fname[FNAME_SIZE];
    int flags = O_WRONLY;

    create_symbolic_file_name(fname);
    create_test_file(fname);

    __assume(__file_exists(fname));

    int fd = open(fname, flags, 0644);
    __assert(open_succeeds(fd));

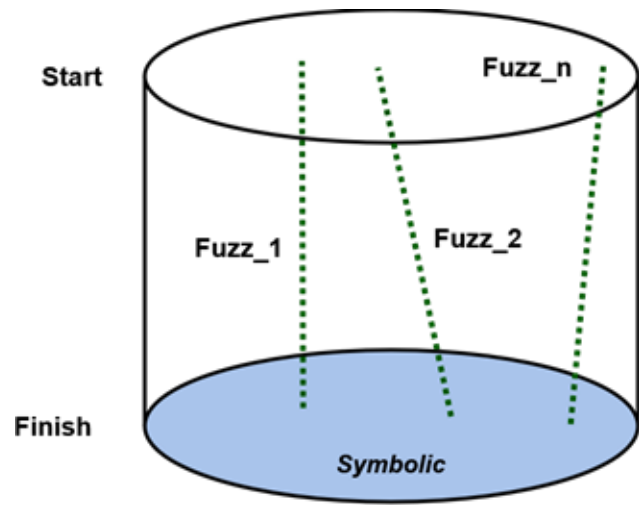
    ssize_t wret = write(fd, "hello", 5);
    __assert(write_all(wret, 5));

    cleanup_fd(fd);
    return 0;
}
```

Validation

Fuzzing file-system summaries

- 1 Run the **summary** with symbolic inputs
- 2 **Seed** inputs from the summary path constraints
- 3 **Fuzz** the real function in the real file system
- 4 Check that the concrete paths **satisfy** the path constraints

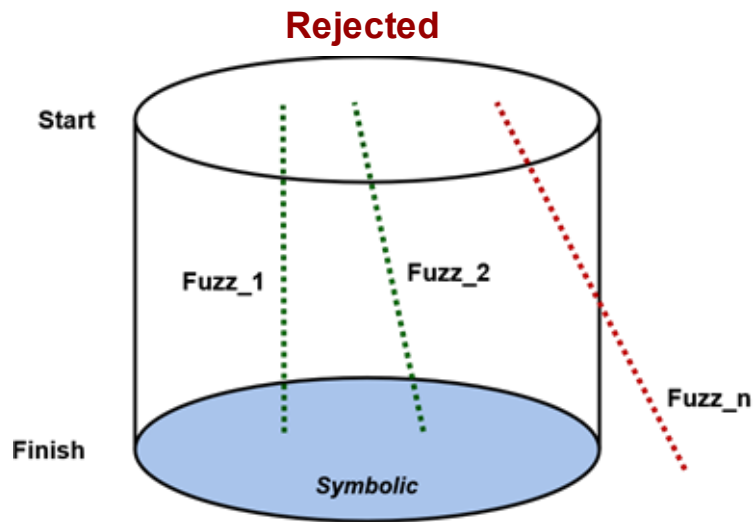


.....
concrete run covered by the summary

Validation

Fuzzing file-system summaries

- 1 Run the **summary** with symbolic inputs
- 2 **Seed** inputs from the summary path constraints
- 3 **Fuzz** the real function in the real file system
- 4 Check that the concrete paths **satisfy** the path constraints



.....
concrete run covered by the summary

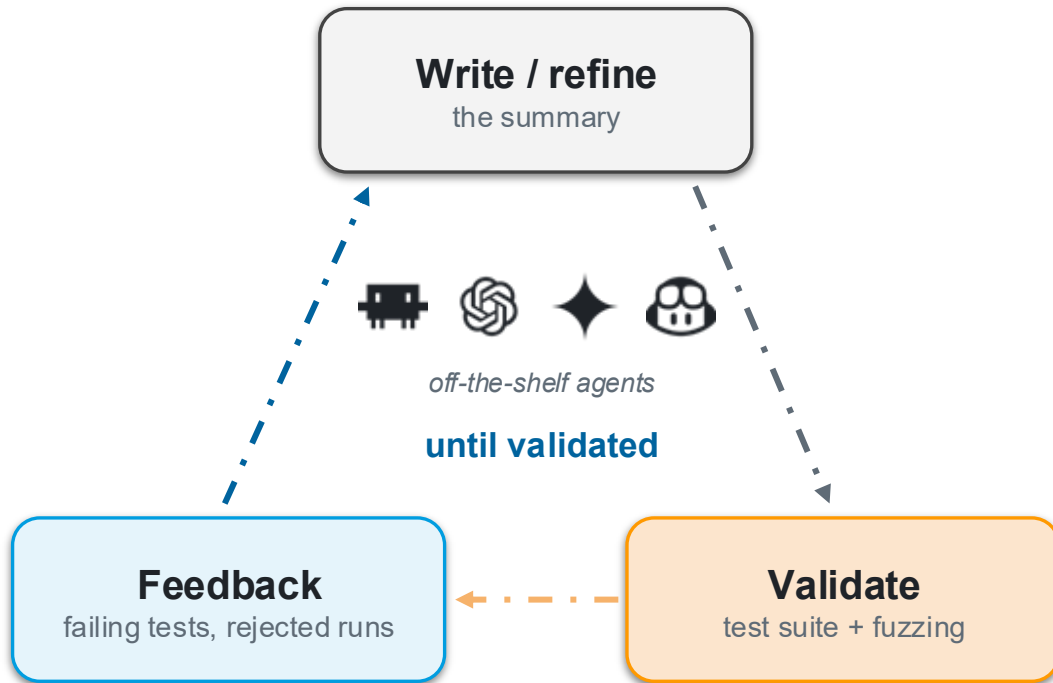
.....
concrete run outside the summary

Summary Generation

The generation loop

What the agent gets:

- libc docs / Natural Language spec
- A description of our API
- Examples of correct summaries
- Typical summarisation patterns
- Our validation tool



Evaluation

Evaluation questions

EQ1 Can our validation pipeline find issues in existing file summaries?

EQ2 Can our methodology generate “correct” summaries?

EQ3 Do our summaries improve on hand-written summaries?

Evaluation

EQ1: Can our validation pipeline find issues in existing file summaries?

4

issues found in KLEE's file summaries

Functions:

open

read

write

All fixed upstream

PRs #1804 and #1825

- 1 Could **write** to a file opened as **read-only**
- 2 Could **read** from a file opened as **write-only**
- 3 `open("A", O_RDONLY)` would override permissions
- 4 `open` calls would silently change permissions

Evaluation

EQ2: Can our methodology generate “correct” summaries?

Generated summaries (so far):

chmod()

close()

dup()

lseek()

open()

read()

write()

Generated by off-the-shelf coding agents

Example: `write()` summary

```
ssize_t summ_write(int fd, const void *buf, size_t count){
    int flags = __file_flags(fd);

    if (flags == -1 || (flags & O_ACCMODE) == O_RDONLY)
        return -1;

    size_t n = __maximize(count);
    __assume(_EQ(count, n));

    if ((flags & O_APPEND) && n > 0){
        ssize_t size = __file_size(fd);
        ssize_t end = __concretize(size);

        while (size != end)
            end = __concretize(size);

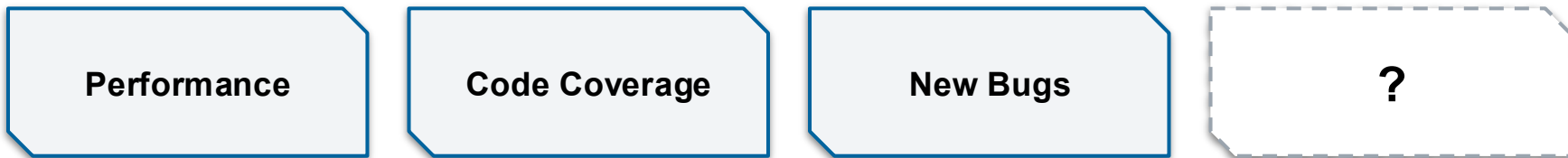
        __file_set_offset(fd, end);
    }

    return __file_write(fd, buf, n);
}
```

Evaluation

EQ3: Do our summaries improve on hand-written summaries?

Work in progress: how to measure this improvement?



Conclusions

1

A symbolic reflection API
with support for file-system
functions

- Implemented in *angr* and *KLEE*

2

A summary validation tool
with support for file
summaries

- `libc` file test suite
- Fuzzing engine

3

An agentic tool for
summary generation

- Off-the-shelf coding agents
- Templates + validation feedback



Thank you!

Feedback? Please reach out