Combining Static Analysis Error Traces with Dynamic Symbolic Execution

Frank Busse • Pritam M. Gharat • Cristian Cadar • Alastair F. Donaldson

3rd International KLEE Workshop on Symbolic Execution
15–16 September 2022, London
Combining Static Analysis Error Traces with Dynamic Symbolic Execution (Experience Paper)

Frank Busse
Imperial College London
London, United Kingdom
fbusse@imperial.ac.uk

Cristian Cadar
Imperial College London
London, United Kingdom
c.cadar@imperial.ac.uk

Prithv Gharat
Imperial College London
London, United Kingdom
prithv.gharat@gmail.com

Alastair F. Donaldson
Imperial College London
London, United Kingdom
alastair.donaldson@imperial.ac.uk

CCS CONCEPTS
• Software and its engineering — Software testing and debugging.

KEYWORDS
Software testing, symbolic execution, static analysis, KLEE, Clang Static Analyzer, Infer

1 INTRODUCTION

Static analysis is a popular method for assuring developers in building correct and secure software. Despite the availability of static analysis tools, e.g. open source tools such as the Clang Static Analyzer [14], Drools [2] and Infer [10], and commercial offerings such as CodeSonar [19], Coverity Scan [16] and Fortify [21], many projects still disregard these tools due to incorrect bug reports, known as false positives. The more time developers waste investigating reports that turn out to be false positives, the more likely they are to abandon using a static analysis tool in the future.

We report our experience designing and evaluating a technique that aims to automate the process of confirming potential bug reports emitted by static analysis. If successful, such a technique could make static analyzers more useful in practice by reducing the amount of time that would need to be spent triaging reports of potential bugs. Given a bug report from a static analysis tool, our idea is to use dynamic symbolic execution (DSE) [9] to try to automatically generate an input that triggers the reported bug. Suppose a static analysis reports a possible bug at a given program location. The analyser typically yields a trace describing (possibly incomplete) details of a path through the program that, if followed, might trigger the bug. Our idea is to then apply a DSE tool to the program, additionally providing the DSE tool with information related to the trace. Rather than attempting to explore all paths of the program in the hope of finding some bug, the DSE tool exploits the trace to explore a manually-pushed subset of paths that agree with the trace, with the aim of confirming the specific
off-the-shelf static analyser

traces

off-the-shelf symbolic executor

concrete inputs for true positives

developers
```c
int main (int argc, char *argv[]) {
    uint8_t in1 = argv[1][0];
    uint8_t in2 = argv[1][1];
    uint8_t in3 = argv[1][2];

    uint8_t *p0, *p1;
    p0 = malloc(sizeof(uint8_t));
    *p0 = in1;

    while (in1 > 'H' - 2) {
        if (in1 == 'H')
            if (in2 == 'i') {
                p1 = p0;
                if (in3 == '!')
                    free(p1);
            }
        --in1;
    }

    int result = *p0;
    free(p0);
    return result;
}
```
Example
Instrumentation

```c
int main (int argc, char *argv[]) {
    uint8_t in1 = argv[1][0];
    uint8_t in2 = argv[1][1];
    uint8_t in3 = argv[1][2];

    uint8_t *p0, *p1;
    p0 = malloc(sizeof(uint8_t));
    *p0 = in1;

    while (in1 > 'H' - 2) {
        if (in1 == 'H')
            if (in2 == 'i') {
                p1 = p0;
                if (in3 == '!')
                    free(p1);
            --in1;
    }

    int result = *p0;
    free(p0);
    return result;
}
```
Targeted Search Heuristic

- drives execution engine **towards** instrumented lines
- **skips** unreachable steps
- **terminates** states that can't reach final step
- **prioritises** states that
  - reached **more steps**
  - are **closer to next step**
The static analysis error traces in our experiments in general do not add (m)any benefits when combined with targeted symbolic execution.