Technische Universität München

Fakultät für Informatik
Lehrstuhl 22
Software Engineering
Prof. Dr. Alexander Pretschner

Supervisor:
Saahil Ognawala
(ognawala@in.tum.de)

Boltzmannstraße 3, 85748
Garching bei München

Tel: +49 89 289 17386

Web: www22.in.tum.de

Fast and Parallel Compositional Analysis of Vulnerabilities with Fuzzing And Targeted Symbolic Execution

Bachelor Thesis

Starting: Immediately

Context

To automatically find vulnerabilities, we mainly consider two techniques – Symbolic execution and fuzzing. Symbolic execution [1] is a powerful way to analyze programs by executing them using "symbolic" values instead of concrete values, and exploring as many paths as possible. It is also an effective way to generate inputs (exploits) that lead to potential vulnerabilities in a program [2]. Fuzzing [3] is a smart variation on random testing that uses a few manually generated inputs (*seed inputs*) and mutates them to trigger previously unseen behaviour in the program. However, symbolic execution suffers from path-explosion and constraint solving issues while fuzzing is infamous for low coverage, because it cannot pass "hard" conditional statements in programs.

Macke [4] is a compositional analysis tool that applies "divide-and-conquer" approach to a program with smaller components that interact based on input pre-conditions and output post-conditions. However, the step of analyzing these smaller components can, currently, be only accomplished using symbolic execution. We would like to extend it with fuzzing.

Goal

The goals of this thesis are as follows –

1. Analyze the codebase of Macke and identify re-usable components.
2. Use an existing library for fuzzing arbitrary functions.
3. Using wrapper functions, similar to Macke wrappers for isolated functions, run parallel fuzzer instances for individual functions in a C program.
4. Using existing helper tools, including targeted symbolic execution, create fuzzing summaries and generate compositional analysis report.
5. Compare effectiveness and efficiency of the approach to pure symbolic execution and the state-of-the-art in Macke.

Workplan

In order to fulfill the above goals, your tasks will include the following –

1. Study symbolic execution and compositional analysis, including their implementations and target-specific techniques.
2. Familiarize yourself with KLEE22 [5] and MACKE, our in-house tools for compositional symbolic execution of C programs.
3. Familiarize yourself with libFuzzer [6], an LLVM plugin for fuzzing standalone libraries, instead of executable binaries.
4. Adapt our in-house tools to automatically compile relevant objects for analysis –
    i. Use LLVM 6.0 with libFuzzer, and
    ii. Use LLVM 3.4 with Macke.

5. Adapt Macke function-isolation functionality to parse function parameters and pass them to libFuzzer.
6. Adapt Macke parallelization feature to run fuzzer in parallel for all functions in the program.
7. If the fuzzer finds any inputs leading to a crash or buffer overflow, parse these inputs and output them in a representation consistent with the existing Macke representation.
8. Adapt Macke to read the crashing inputs and run phase-2 analysis (technology already exists in Macke) with them.
9. Integrate the above steps completely into Macke.
10. Evaluate your implementation on an existing benchmark set of C programs and compare the effectiveness and efficiency w.r.t. old Macke and KLEE.
11. Record your studies, design, evaluation criteria and findings in a thesis document in a systematic manner.

Fakultät für Informatik
Lehrstuhl 22
Software Engineering
Prof. Dr. Alexander Pretschner

Supervisor:
Saahil Ognawala
(ognawala@in.tum.de)

Boltzmannstraße 3, 85748
Garching bei München

Tel: +49 89 289 17386

Web: www22.in.tum.de

Eligibility

An ideal candidate should have some *programming experience in Python and C++*.

We are interested in highly motivated students for this study, who will help further our research in the field of automatic analysis of programs for security vulnerabilities. We expect the student to produce high-quality deliverables, both in terms of scientific and engineering contributions.

References

[1] King, J. C. "Symbolic execution and program testing." 1976.
[2] Cadar, C., et al. "EXE: A system for automatically generating inputs of death using symbolic execution." 2006.
[3] Sutton, M., et al. “Fuzzing: brute force vulnerability discovery”. 2007.
[4] Ognawala, S., et al. "MACKE: Compositional analysis of low-level vulnerabilities with symbolic execution." 2016.
[5] KLEE22, https://github.com/tum-i22/klee22
[6] LibFuzzer, https://llvm.org/docs/LibFuzzer.html