UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Static Symbolic Execution in SVF

by Guanqin ZHANG

Thesis submitted in fulfilment of the requirements for the degree of

Master of Analytics (Research) in Computer Science

under the supervision of Yulei SUI

University of Technology Sydney
Faculty of Engineering and Information Technology

Jan 2022

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, *Guanqin ZHANG* declare that this thesis, is submitted in fulfilment of the requirements for the award of *Master of Analytics (Research) in Computer Science* in the *Faculty of Engineering and Information* at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature: Production Note: Signature removed prior to publication.

Date: Jan 25th, 2022

Abstract

This thesis presents the implementation of a static symbolic execution (**SSE**) tool on top of SVF framework. SSE is a well-studied technique for program testing and verification, which is based on collecting the program execution paths and exploring the program behaviours in the form of the constraints. SSE replies on SVF-IR language to translate the program into logical formulas of a particular rules-based form which supplies useful program properties for use in the program analysis. SSE maintains all elements of program running data and its manipulations. More precisely, SSE traverses on program control flow graph and modulates program behaviours, e.g., buffer-over-flow is an anomaly in program, which can be identified that the data overruns the buffer's boundary and overwrites adjacent memory locations. Experiments conducting our implementation of SSE indicates many benefits by using symbolic execution, e.g., depth-first searching on particular paths to mimic real program execution process, or finding certain program anomalies based on translating rules.

Table of Contents

List of Figures

List of Tables