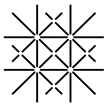

University of Basel

Optimizing Symbolic Execution Through Taint Analysis and Path Prioritization

Bachelor thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Databases and Information Systems (DBIS) Group
https://dbis.dmi.unibas.ch/

Examiner: Dr. Marco Vogt
Supervisor: Prof. Dr. Christopher Scherb

Ruben Hutter
ruben.hutter@unibas.ch
2020-065-934

02.07.2025

Acknowledgments

I would like to express my sincere gratitude to Prof. Dr. Christopher Scherb for his supervision and guidance throughout this thesis. His expertise in program analysis and symbolic execution provided essential direction for this research.

I am grateful to Dr. Marco Vogt for his role as examiner and for facilitating the opportunity to pursue this research topic. His feedback and suggestions helped refine both the technical approach and the presentation of this work.

I would like to thank Ivan Giangreco for providing the LaTeX thesis template used for this document, which greatly facilitated the formatting and structure of this work.

I also acknowledge my fellow student Nico Bachmann for developing the Schnauzer visualization library, which enhanced the presentation and analysis of the results in this work.

I thank my family and friends for their encouragement and support during my studies, which made completing this thesis possible.

Finally, I acknowledge the developers of the angr binary analysis framework, whose comprehensive platform enabled the implementation of the techniques presented in this work.

Abstract

Symbolic execution is a powerful program analysis technique widely used for vulnerability discovery and test case generation. However, its practical application is often hampered by scalability issues, primarily due to the "path explosion problem" where the number of possible execution paths grows exponentially with program complexity. This thesis addresses this fundamental challenge by proposing an optimized approach to symbolic execution that integrates taint analysis and path prioritization.

The core contribution is a novel exploration strategy that moves away from uniform path exploration towards targeted analysis of security-critical program behaviors. The approach prioritizes execution paths originating from memory allocations and user input processing points, as these represent common sources of vulnerabilities. By leveraging dynamic taint analysis, the system identifies and tracks data flow from these critical sources, enabling the symbolic execution engine to focus computational resources on paths influenced by tainted data while deprioritizing paths with no dependency on external inputs.

The implementation integrates this taint-guided exploration strategy with the angr symbolic execution framework, introducing a scoring mechanism that dynamically adjusts path prioritization based on taint propagation. The effectiveness of this optimization is evaluated through comparative analysis, examining runtime efficiency, path coverage quality, and vulnerability discovery capabilities. Results demonstrate that this approach can significantly reduce the search space while maintaining or improving the detection of security-relevant program behaviors, making symbolic execution more practical for large and complex software systems.

Table of Contents

1 Introduction

Symbolic execution has emerged as one of the most powerful techniques for automated program analysis, offering significant advantages over traditional testing methods for vulnerability discovery and test case generation. By representing program inputs as symbolic variables rather than concrete values, symbolic execution engines can systematically explore multiple execution paths within a single analysis run, potentially uncovering bugs that would be difficult to find through conventional testing approaches.

Despite its theoretical promise, symbolic execution faces a fundamental scalability challenge known as the "path explosion problem." As program complexity increases, the number of possible execution paths grows exponentially, quickly overwhelming computational resources and rendering the analysis intractable for real-world software systems. This limitation has been a persistent barrier to the widespread adoption of symbolic execution in practical security analysis workflows.

Current symbolic execution engines typically employ uniform exploration strategies, treating all program paths with equal priority regardless of their potential security relevance. This approach wastes significant computational resources on code paths that are unlikely to contain vulnerabilities, while potentially missing critical security-sensitive execution flows. For instance, paths that process user-controlled input or handle memory allocations are statistically more likely to contain exploitable vulnerabilities than paths that perform simple arithmetic operations or access read-only data structures.

This thesis addresses the path explosion problem by proposing a novel approach that integrates taint analysis with symbolic execution to enable intelligent path prioritization. The core insight is that not all execution paths are equally important from a security perspective. By leveraging taint analysis to identify and track data flow from critical sources such as user inputs and memory allocation sites, we can guide the symbolic execution engine to focus its computational resources on paths most likely to exhibit security-relevant behaviors.

The proposed approach introduces a dynamic scoring mechanism that continuously evaluates the security relevance of execution states based on taint propagation patterns. States that process tainted data receive higher priority scores, while states operating exclusively on untainted data are deprioritized or pruned entirely. This selective exploration strategy aims to maintain the thoroughness of symbolic execution for security-critical code while dramat-

ically reducing the analysis time by avoiding exhaustive exploration of irrelevant program regions.

The main contributions of this work are:

Taint-Guided Path Prioritization: A novel integration of dynamic taint analysis with symbolic execution that uses taint propagation patterns to intelligently prioritize exploration of security-relevant execution paths.

Adaptive Scoring Algorithm: A scoring mechanism that dynamically adjusts path priorities based on real-time taint analysis results, enabling the symbolic execution engine to focus computational resources on the most promising program regions.

Practical Implementation: A complete implementation of the proposed approach using the angr symbolic execution framework, demonstrating the feasibility and effectiveness of taint-guided exploration in a production-quality tool.

Empirical Evaluation: Comprehensive evaluation comparing the proposed approach against standard symbolic execution techniques, measuring improvements in analysis efficiency, vulnerability discovery rate, and overall scalability.

The effectiveness of this optimization is evaluated through extensive experimentation on representative programs, examining key metrics including runtime efficiency, path coverage quality, and vulnerability detection capabilities. Results demonstrate that the taint-guided approach can significantly reduce analysis time while maintaining or improving the detection of security-relevant program behaviors, making symbolic execution more practical for analyzing large and complex software systems.

The remainder of this thesis is organized as follows. Chapter 2 provides essential background on symbolic execution, taint analysis, and the angr framework that forms the foundation for this work. Chapter 3 presents the conceptual framework and theoretical algorithms underlying the taint-guided exploration strategy. Chapter 4 details the practical implementation, including integration with angr and the design of the scoring mechanism. Chapter 5 presents a comprehensive evaluation of the approach, comparing its performance against standard symbolic execution techniques. Chapter 6 discusses related work in symbolic execution optimization and path prioritization. Chapter 7 concludes with a summary of contributions and implications for future research. Chapter 8 explores potential extensions and future research directions, while Chapter 9 documents the use of AI-assisted technologies in the development of this thesis.

2
Background

Base for the thesis, definitions, concepts, and related work (similar approaches or works that give a background for the thesis).

3

Improving Symbolic Execution ...

This is a short conclusion on the thesis template documentation. If you have any comments or suggestions for improving the template, if you find any bugs or problems, please contact me.

How does it work conceptually? (not implementation) which path i choose and why. limiter -¿ je weiter unter desto schwieriger dass ich eine vulnerability triggere Pseudo code of the algorithm

4

Practical Implementation

This is a short conclusion on the thesis template documentation. If you have any comments or suggestions for improving the template, if you find any bugs or problems, please contact me.

How does the script work? Implementation details, how to use it, how to run it, how to set up the environment. Strategies that I use, LoopSeer...

5
Evaluation

Compare my work to default angr strategy and other tools. (Maybe I will implement a benchmark program to compare the performance of my tool with angr and other tools.)

6
Related Work

Evtl. already mentioned in background chapter

Here other approaches to symbolic execution, like the Paper: "MACKE: Compositional Analysis of Low-Level Vulnerabilities with Symbolic Execution"

7
Conclusion

Rückblick auf die Arbeit, was wurde erreicht (evtl. merge with Future Work)

8
Future Work

Some ideas for future work could be: - Change meta file to actual header file - Make it work also for ARM and X86 (checking stack and heap arguments) - Check that it works also for libraries (not only for main function) - Let the script analyze a complex program (multiple files) and get an output over all (now it only works for one file at a time)

9
Usage of AI

Used AI tools, like ChatGPT, to help with the writing of this thesis. I used it to generate ideas, to write parts of the text, to improve the text, and to check the grammar and spelling. I also used it to generate code snippets and to explain code snippets.

Bibliography

A

Appendix

University of Basel

Faculty of Science

Declaration on Scientific Integrity

(including a Declaration on Plagiarism and Fraud)
Translation from German original

Title of Thesis:

Name Assessor: _____

Name Student: _____

Matriculation No.: _____

I attest with my signature that I have written this work independently and without outside help. I also attest that the information concerning the sources used in this work is true and complete in every respect. All sources that have been quoted or paraphrased have been marked accordingly.

Additionally, I affirm that any text passages written with the help of AI-supported technology are marked as such, including a reference to the AI-supported program used. This paper may be checked for plagiarism and use of AI-supported technology using the appropriate software. I understand that unethical conduct may lead to a grade of 1 or "fail" or expulsion from the study program.

Place, Date: _____ Student: _____

Will this work, or parts of it, be published?

No

Yes. With my signature I confirm that I agree to a publication of the work (print/digital) in the library, on the research database of the University of Basel and/or on the document server of the department. Likewise, I agree to the bibliographic reference in the catalog SLSP (Swiss Library Service Platform). (cross out as applicable)

Publication as of: _____

Place, Date: _____ Student: _____

Place, Date: _____ Assessor: _____

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis.