TraceGuard: Taint-Guided Symbolic Execution

Bachelor Thesis Defense

Ruben Hutter

July 8, 2025

University of Basel, Faculty of Science
Department of Mathematics and Computer Science

Introduction

The Path Explosion Problem

- Symbolic execution explores all possible program paths
- Number of paths grows exponentially with program complexity
- Traditional approaches treat all paths equally
- Security vulnerabilities often occur in specific paths
- Need for intelligent path prioritization

[Path Explosion Diagram]

Research Motivation

Problem Statement

- Symbolic execution suffers from exponential path explosion
- Uniform exploration wastes resources on irrelevant paths
- Security analysis requires focus on user-input processing paths

Solution Approach

- Integrate taint analysis with symbolic execution
- Prioritize paths that process potentially malicious data
- Guide exploration toward security-relevant code regions

Background

Symbolic Execution

- Program analysis technique using symbolic variables
- Explores multiple execution paths simultaneously
- Generates constraints for path conditions
- Powerful for vulnerability discovery but suffers from scalability issues

Key Challenge: Path explosion makes exhaustive analysis intractable

Taint Analysis

- Tracks data flow from untrusted sources (taint sources)
- Monitors propagation through program operations
- Identifies when tainted data reaches critical operations (sinks)
- Provides security-focused view of program behavior

Key Insight: Security vulnerabilities are more likely in paths processing tainted data

TraceGuard Approach

Core Methodology

1. **Taint Source Recognition:** Hook input functions (fgets, scanf, read)
2. **Dynamic Taint Tracking:** Monitor data flow through symbolic execution
3. **Path Prioritization:** Score states based on taint interaction
4. **Guided Exploration:** Focus resources on high-priority paths

Result

Transform uniform exploration into security-focused analysis

Implementation Architecture

Core Components:

- TraceGuard Class
- Function Hooking System
- Taint Tracking Engine
- Custom Exploration Technique
- Visualization Integration

Built on Angr Framework:

- Binary analysis platform
- Symbolic execution engine
- Multi-architecture support
- Extensible Python interface

Evaluation

Experimental Setup

Benchmark Suite

- 7 synthetic test programs with known vulnerabilities
- Programs designed to challenge symbolic execution
- Multiple runs with 120-second timeout per execution
- Comparison: TraceGuard vs Classical Angr

Metrics

- Vulnerability detection rate
- Execution time performance
- Basic block coverage efficiency
- State exploration patterns

Key Results

Vulnerability Detection:

- 100% detection rate across all tests
- 5× improvement in challenging scenarios
- Consistent performance across multiple runs

[Results Visualization]

Efficiency:

- Competitive execution times
- 36.8% to 75.0% of classical coverage
- Focused exploration strategy

Contributions

Research Contributions

Novel Integration

- First comprehensive framework for real-time taint-guided symbolic execution
- Dynamic state prioritization based on security relevance
- Practical implementation demonstrating feasibility

Technical Achievements

- Custom Angr exploration technique
- Function-level taint tracking system
- Adaptive scoring algorithm with configurable thresholds
- Comprehensive benchmarking infrastructure

Future Work

Limitations and Future Directions

Current Limitations:

- Evaluation limited to synthetic test programs
- Primary focus on AMD64 C/C++ binaries
- Dependency on accurate taint source identification

Future Enhancements:

- Real-world application validation
- Multi-architecture and language support
- Enhanced taint granularity (byte-level tracking)
- Integration with fuzzing frameworks
- Machine learning-guided exploration strategies

Conclusion

Conclusion

Achievements

- Successfully integrated taint analysis with symbolic execution
- Demonstrated significant improvements in vulnerability discovery
- Maintained competitive performance while reducing exploration scope
- Provided foundation for security-aware program analysis

Impact: TraceGuard transforms symbolic execution from uniform exploration into intelligent, security-focused analysis, addressing fundamental scalability challenges while improving vulnerability detection effectiveness.

Questions?

Thank You

Questions and Discussion

TraceGuard: Taint-Guided Symbolic Execution
for Enhanced Binary Analysis

Ruben Hutter
University of Basel