Comparing Static and Dynamic Taint Analysis: Exploring Input Validation in Java

G.G. MAMATSASHVILI, MICHELE BANDINI, MARKUS SKOV NIELSEN, MATHIAS GILBERT, EKOW DADZIE, and FRANCESCO FAGGIANO, DTU Compute - Group 18, Denmark

Motivation. Cyber-attacks have become increasingly frequent and sophisticated, becoming a major concern for online services providers. Among the most common and dangerous vulnerabilities are injection attacks, in which untrusted user inputs are used to compromise systems. Preventing these attacks is challenging as such vulnerabilities are often hidden deep within large code structures.

In order to train for security, deliberately vulnerable applications have been built, such as *OWASP WebGoat*. Their complexity mirrors real-world scenarios, making it an ideal target for testing advanced program analysis techniques. However, current methods often struggle with precision and recall. This is where taint analysis has emerged as a promising approach.

Taint analysis, a specialized form of data flow analysis, tracks the flow of untrusted data through a program to identify when it reaches unauthorized sensitive operations. However, the effectiveness of taint analysis can vary significantly depending on whether a static or dynamic approach is employed. Static analysis examines the program's source code without execution, potentially offering comprehensive coverage but risking false positives. Dynamic analysis, on the other hand, observes the program during runtime, providing precise results for executed paths but potentially missing vulnerabilities in unexecuted code.

Understanding the relative strengths and limitations of these program analysis approaches is crucial for developing more effective vulnerability detection techniques. This research aims to contribute to this understanding by comparing static and dynamic taint analysis methods in their ability to detect input vulnerabilities accurately and efficiently.

Research Question. Does the static taint analysis outperform the dynamic taint analysis in detecting SQL injection and command execution vulnerabilities in Java web applications, with respect to precision, recall, and performance?

Approach. This project aims to compare the efficacy of static and dynamic analysis techniques in detecting input validation vulnerabilities through taint analysis. Two teams, each comprising three members, will conduct parallel investigations using static and dynamic analysis methods, respectively. The target language for this research is Java.

The scope of the analysis encompasses common web application vulnerabilities, focusing on two primary input sources: standard form inputs and HTTP parameters. The sensitive sinks under scrutiny are SQL queries and system command executions, both of which are frequent targets for malicious exploitation. Code will be considered adequately sanitized if it can be demonstrated that SQL injections and unintended command executions are no longer feasible.

For static analysis, the team will utilize Soot, a widely-recognized framework for analyzing Java bytecode. Tainted data will be represented as a set of variable names, facilitating efficient tracking

Authors' Contact Information: G.G. Mamatsashvili, s240055@student.dtu.dk; Michele BANDINI, s243121@student.dtu.dk; Markus Skov NIELSEN, s214938@student.dtu.dk; Mathias GILBERT, s243250@student.dtu.dk; Ekow DADZIE, s240652@student.dtu.dk; Francesco FAGGIANO, s232436@student.dtu.dk, DTU Compute - Group 18, Kgs Lyngby, Denmark.

and analysis. Control flow graphs (CFGs) will be constructed to aid in data flow analysis, allowing for a comprehensive examination of potential taint propagation paths within methods.

The dynamic analysis team will employ a debugger-based approach, instrumenting the Java Virtual Machine (JVM) to monitor execution in real-time. Tainted data will be tracked by attaching metadata tags to variables and objects as they flow through the program. To ensure adequate path coverage, the team will implement a fuzzing strategy, generating diverse inputs to explore various execution paths and uncover potential vulnerabilities that may not be immediately apparent through static inspection.

To ensure a consistent and realistic testing environment, both teams will conduct their analyses on OWASP WebGoat, a deliberately vulnerable Java-based web application designed for security training. WebGoat provides a complex, real-world-like codebase that includes various injection vulnerabilities, making it an ideal candidate for this comparative study. By using WebGoat, we can evaluate the effectiveness of both static and dynamic taint analysis approaches in detecting known vulnerabilities within a controlled yet realistic setting. Both methods will be evaluated against a common set of benchmarks, including known vulnerable code samples and real-world applications. The comparative analysis will consider several key metrics: *detection rate of known vulnerabilities, false positive and false negative rates, precision and recall, analysis runtime and memory usage.*

The final report will present a detailed comparison of the strengths and weaknesses of both static and dynamic approaches in the context of taint analysis. It will discuss scenarios where each method might be most effective and explore potential synergies between the two approaches. The findings aim to contribute to the ongoing discourse on effective vulnerability detection strategies and inform best practices for secure software development in Java-based environments.

Evaluation Strategy. To evaluate and compare the performance of the analyses they will be tested against specific test cases in the OWASP Benchmark. The analyses will be tested against specific test cases in OWASP WebGoat, focusing particularly on the modules related to SQL injection ('sqli') and command injection ('cmdi'). This will allow us to evaluate the effectiveness of each approach in detecting these specific types of vulnerabilities within a complex, realistic application environment.

In this context they will be evaluated based on a number of metrics:

- **Performance**, the time taken for each analysis to execute. This would give a reasonable overview on the practicality and scalability of the analysis for specific test cases. Additionally, knowing the performance helps identify slow points in the analysis process and allows for improvements to methods for larger codebases.
- **Correctness**, in vulnerability detection there is a focus on how accurately each analysis identifies actual vulnerabilities and avoids false positives or false negatives. It ensures that the analysis is both reliable and actionable, reducing unnecessary debugging for false alarms and as well as the risk of leaving flaws undetected. Achieving the right balance between precision (avoiding false positives) and recall (minimizing false negatives) is key to the effectiveness of any vulnerability detection strategy.

Project Plan. For implementing the analyses the group will split into two teams of three, one working on static analysis and the other on dynamic analysis. The two teams will work in parallel with weekly meetings updating each other on the progress. A week to week plan for the two teams can be found in appendix A.

A Project Plan

Week	Static Analysis Team	Dynamic Analysis Team
1	Set up environments, familiarize with WebGoat, finalize plan, assign roles, establish communication and benchmarking protocols	
2-3	Implement static taint analysis (Soot), develop custom rules (SQL injection, command execution), test on vulnerable code samples	Set up Java debugger, implement taint tracking, fuzzing, test on vulnerable code samples
4	Run static and dynamic analysis on WebGoat, collect metrics (detection rates, false positives/negatives, performance)	
5	Analyze findings, optimize techniques, re-run static and dynamic analysis on WebGoat	
6	Compile final results, compare static vs dynamic approaches, finalize paper	

Table 1. Project Plan