Institute for Software Integrated Systems

Technical Report

TR#: **ISIS-17-104**

Title: **Janalyzer: A Static Analysis Tool for Java Bytecode**

Authors: **Daniel Balasubramanian, Zhenkai Zhang, Dan McDermet and Gabor Karsai**

Janalyzer: A Static Analysis Tool for Java Bytecode

Daniel Balasubramanian, Zhenkai Zhang, Dan McDermet, Gabor Karsai
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37212
Email: {daniel.a.balasubramanian, zhenkai.zhang, dan.mcdermet, gabor.karsai}@vanderbilt.edu

Abstract

Static analysis attempts to discover program properties and answer questions about a program's potential behavior without actually running the program. Example properties include things such as loop bounds, reachability and taint analysis. Even though there is a vast body of literature describing program various program analysis techniques and algorithms, implementing these algorithms can be challenging. Understanding how to make the techniques scale to real-world programs is an additional challenge.

This technical report describes our work implementing several static analysis techniques on Java bytecode in a tool we call the Janalyzer. We describe the algorithms we used along with the trade-offs we made in terms of scalability and precision.

Keywords

Static analysis; program analysis; worst-case execution; side-channel analysis; bytecode

I. Introduction

Static program analysis is an active area of research with a vast body of literature describing many techniques and algorithms to statically extract information about a program and its possible behaviors. In fact, the literature is so large that even knowing where to start can be a challenge.

Another big challenge is understanding the trade-offs between scalability and precision. For instance, static program slicing can have difficulty scaling while remaining precise enough to give useful information on large, real-world programs [1]. In order to make many algorithms and techniques usable on complex programs, some sort of approximation is introduced, which is often not described in the research literature.

We have implemented several static analysis techniques and algorithms in a tool we call the "Janalyzer", short for "Java Analyzer". The Janalyzer takes as input a Java bytecode program, performs its analyses, and then presents these to the user in a graphical user interface. The user can then view the analysis results, as well as make additional queries.

II. Program analysis components

The Janalyzer implements several different types of analysis for Java bytecode programs. These include the following.

- Call graph generation. This basic information tells which methods each method can call. This forms the basis for many of the other analyses.
- Control-flow graph generation. This describes the flow of control within a single method.
- Inter-procedural nested loop identification. This describes how loops are nested across method calls.
- Program slicing. This tells which program statements are affected by another program statement.
- Suspicious loop identification. This identifies which loops are "suspicious", where suspicious is defined to mean that the loop exit condition depends on an object that is modified inside the body of the loop itself.
- Unbalanced branch identification. This describes which branches may be unbalanced with respect to some metric, such as the number of instructions executed.

References

[1] M. Sridharan, S. J. Fink, and R. Bodík, "Thin slicing," in *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2007.