Scalable Static Analysis to Detect Security Vulnerabilities: Challenges and Solutions

Nathan Keynes, François Gauthier, Nicholas Allen, Diane Corney, Padmanabhan Krishnan, Cristina Cifuentes

{nathan.keynes, francois.gauthier, nicholas.allen, diane.corney, paddy.krishan, cristina.cifuentes}@oracle.com
Oracle Labs

Abstract

Parfait [1] is a static analysis tool originally developed to find defects in C/C++ systems code. It has since been extended to detect injection attacks [3] in Java and PL/SQL[1] applications. Parfait has been deployed at Oracle, is used by thousands of developers, and can be integrated at commit- or build-time. This poster presents some of the challenges we encountered while extending Parfait from a defect analyser for C/C++ to a security analyser for Java and PL/SQL, and the solutions that enabled us to analyse a variety of commercial enterprise applications in a fast and precise way.

1. Precision

Parfait's focus has always been on mitigating risk and generating true positive reports rather than achieving soundness. For our C/C++ analysis, the key to precision was primarily a combination of path-sensitivity, field-sensitivity, and precise tracking of concrete values. For Java applications, however, our experience suggests that precise call-graphs and field-sensitive analysis [2] are key to achieving > 90% overall precision. In practice, this means that Parfait will often aggressively under-approximate call targets or field aliases when it encounters constructs like reflection, dependency injection, or network routing, and complement its analysis with code models instead.

2. Multiple Languages

All languages supported by Parfait share the same IR, and translators must generate meta-data (e.g. Java class hierarchy) to enable analysis (see Fig. 1).

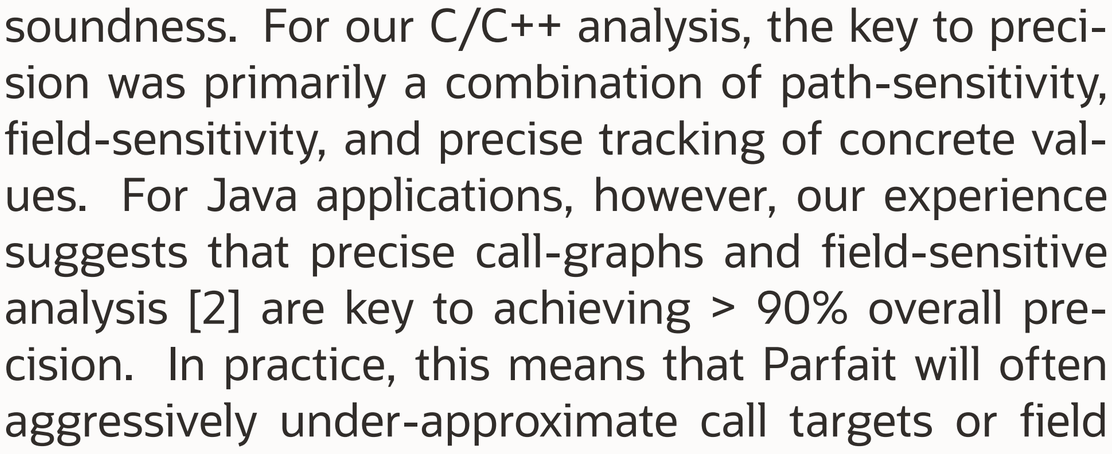

Fig. 1: All languages share the same IR (LLVM bitcode) in Parfait

[1]Java and PL/SQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

3. Novelty: Incremental Analysis

Achieving commit-time analysis on monolithic Java applications of ∼100MLOC is challenging. Because these applications cannot be fully loaded into memory at once, Parfait first scans each function for type and call information to generate a call graph. Our bottom-up summarisation approach can then identify and re-analyse changed methods only, and propagate the new summaries up the call-stack (see: Fig. 2).

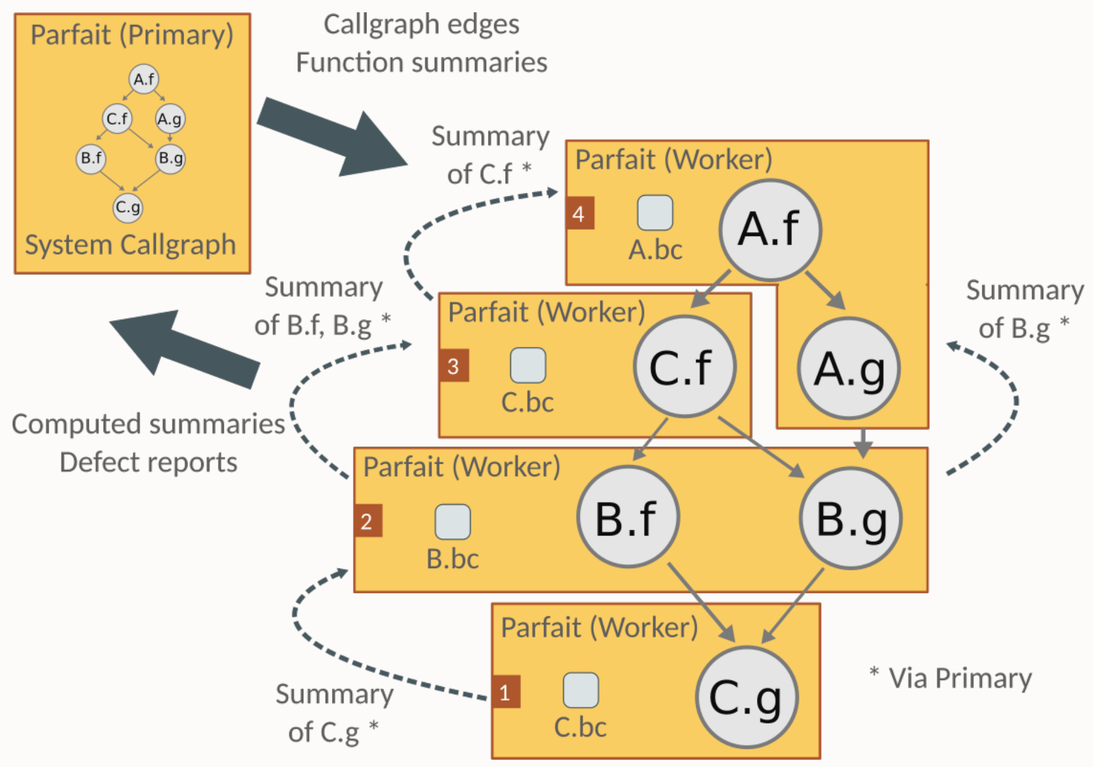

Fig. 2: Inter-module analysis in Parfait

Outcome and Open challenges

Parfait is used by 1000+ Oracle developers on a daily basis. Deploying incremental analysis has brought security closer to developers by enabling commit-time feedback. We plan to address these challenges next:

Cross-language analysis: We are exploring heap abstractions to capture common cross-language taint flows (e.g. code→DB→code) in applications.
Code models: We are exploring ML approaches to create and maintain models of hard-to-analyse code and reduce manual effort to a minimum.

References

[1] C. Cifuentes, N. Keynes, L. Li, N. Hawes, and M. Valdiviezo. Transitioning Parfait into a development tool. *IEEE Security and Privacy*, 2012.

[2] J. Lerch, J. Späth, E. Bodden, and M. Mezini. Access-path abstraction: Scaling field-sensitive data-flow analysis with unbounded access paths. In *ASE*, pages 619–629, 2015.

[3] OWASP. Owasp top ten project. `https://owasp.org/www-project-top-ten/`, Last accessed: 27 January 2022.