Improving the security of infrastructure software using Coccinelle

Lawall, Julia; Hansen, René Rydhof; Palix, Nicolas Jean-Michel; Muller, Gilles

Published in:
ERCIM News

Publication date:
2010

Document version
Publisher's PDF, also known as Version of record

Improving the Security of Infrastructure Software using Coccinelle

by Julia Lawall, René Rydhof Hansen, Nicolas Palix and Gilles Muller

Finding and fixing programming errors in deployed software is often a slow, painstaking, and expensive process. In order to minimise this problem, static analysis is increasingly being adopted as a way to find programming errors before the software application is released. Coccinelle is a program matching and transformation tool that makes it easy for developers to express static analysis-based software defect-finding rules and scan software source code for potential defects.

Defects are continually found in end-user software, even in seemingly reliable software on which millions of people depend. Finding and fixing defects is slow and painstaking: a tester or user finds that the software crashes, that person submits a bug report, a maintainer studies the code to find the root cause of the reported problem, and finally some change is made to fix it. The process then repeats when the next crash occurs. An alternative is to use static analysis, which is being increasingly adopted in both commercial and research tools. In this approach, a tool scans the software source code according to a collection of rules, and signals apparent defects. Static analysis is pervasive: all of the source code is checked, even source code that is rarely executed. Furthermore, defects are often reported at or near the place where the code needs to be changed; it is not necessary to connect an external run-time behavior to a particular source code element.

A static analysis approach, however, is only as good as the set of rules that it checks. Indeed, while the standard defect detection method relies on observed run-time problems to initiate the defect-finding process, static analysis requires that a rule developer anticipate causes, and ideally solutions, of potential problems. What is needed is an approach to easily turn a maintainer's intuition about a potential problem, obtained from either debugging a runtime error or from studying the reports from a static analysis tool, into a rule that can be used to scan the software for similar problems.

During the past four years, supported in part by the Danish Research Council for Technology and Production at the Universities of Copenhagen and Aalborg in Denmark and the French National Research Agency at INRIA in France, we have been developing the Coccinelle program matching and transformation system. Coccinelle allows software developers to express rules as patterns described in terms of source code elements that can be abstracted such that they match not just one specific piece of code, but also similar code structures throughout a software project. Coccinelle patterns can furthermore be annotated using '-' and '+', following the commonly used patch syntax, to indicate code to remove and add, respectively. Thus, Coccinelle rules may not only find defects, but can also fix them. We have used Coccinelle to help in finding and fixing hundreds of defects in the Linux operating system. A typical example of a rule specification used in the context of Linux is shown in the figure. Coccinelle has furthermore been adopted by a number of developers outside of our research group, for use on both open and closed source software projects.

Our most recent work has lead in two new directions to better support the defect-finding process. First, to improve the robustness of a software system, it is necessary to understand how defects are introduced. While Coccinelle enables finding defects in multiple versions of a software project, it does not distinguish between defects that are long lasting and those that are continually fixed and reappear. To obtain a more complete picture, we have developed Herodotos, which correlates defect reports across multiple versions, even in the presence of other changes in the source code. A second direction is the integration of Coccinelle and Clang, the C language frontend developed as part of the LLVM (Low Level Virtual Machine) compiler framework. By integrating Clang, Coccinelle is able to leverage the comprehensive program analysis capabilities found in both Clang and the underlying LLVM framework. This results in better precision for Coccinelle searches, and enables supporting a wider range of searches.

Coccinelle is open source and can be downloaded freely from the Coccinelle web page.

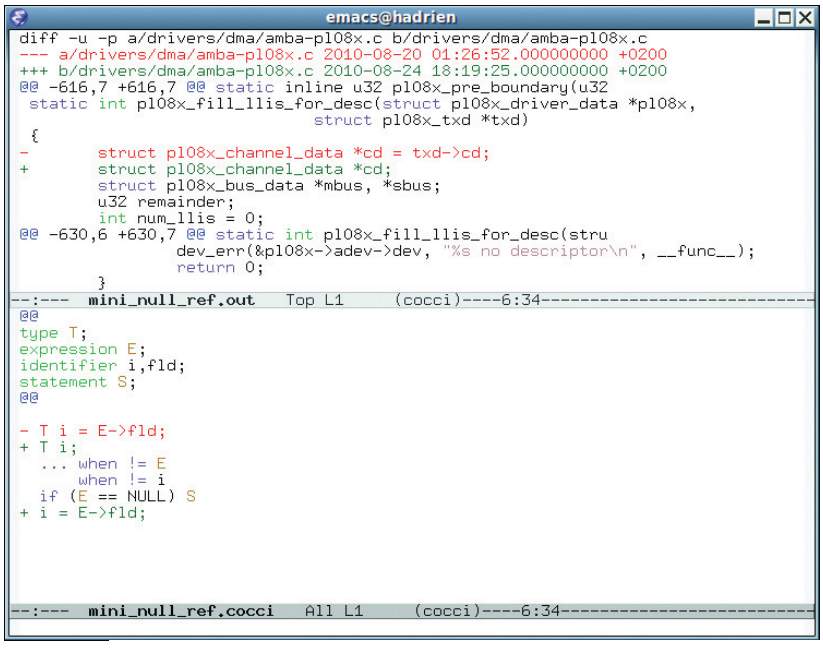

Figure 1: Rule specification and matches in Coccinelle.

Links:
Coccinelle: http://coccinelle.lip6.fr/
Herodotos: http://coccinelle.lip6.fr/herodotos.html
Low Level Virtual Machine (LLVM): http://www.llvm.org/
Clang: http://clang.llvm.org/

Please contact:
Julia L. Lawall
University of Copenhagen / DANAIM, Denmark
Tel: +45 35321405
E-mail: julia@diku.dk