



## Review: Automated Code Scanners

Posted by Justin Schuh on April 13, 2007



We brought three popular static source-code analyzers into our Chicago Neohapsis Real-World Labs®: Fortify [Source Code](#) Analysis (SCA) Suite 4.0, Ounce Labs' Ounce 4.1, and Klocwork K7 7.5. Coverity declined to send us its Prevent analyzer.

We approached these products from the perspective of a development team, focusing on the top five features that most directly relate to the successful detection and remediation of vulnerabilities, and that most affect the development team's productivity. Foremost is the breadth of languages, platforms and development environments supported. If you have applications in multiple languages, you'll want a single analyzer to cover them all.


Fortify's SCA was a standout here, with the broadest range of technology support across the board. SCA's expansive [platform](#) support includes IBM AIX, Linux, Microsoft Windows, Sun Solaris and Mac [OS X](#), primarily due to Fortify's use of the Eclipse [IDE](#) as a base platform, though it fully supports Visual Studio 2003 and 2005. Fortify SCA's real technology advantage becomes apparent in the wide array of languages supported by the analyzer; these include Java, JSP, C/C++, ASP.Net (C# and VB.Net), [SQL](#) procedural languages (TSQL and PLSQL), and XML. Fortify was the only product we tested to provide such broad language support, including support for data-tier languages like SQL.

Ounce Labs' Ounce also has an impressive range, supporting analysis on AIX, Linux, [Windows](#) and Solaris platforms and integration with Visual Studio 2003 and 2005, along with Eclipse and its derivatives, including IBM's Rational environment. Although its language roster isn't as broad as SCA's, Ounce still includes an impressive lineup of Java, JSP, C/C++, and ASP.Net (C# and VB.Net).

Klocwork K7 is built on Eclipse and also covers an array of development platforms. Supported environments include embedded systems IDEs such as QNX Software Systems' Momentics and Wind River's Workbench, and common editors like Emacs, GVim, KDevelop, MetroWerks CodeWarrior, Microsoft Platform Builder and Visual SlickEdit. However, K7's Visual Studio integration stops at 2003; that's a major problem for Visual Studio 2005 shops, but it should be addressed in the 7.7 version. Language support covers C/C++ and Java/JSP only, with a notable absence of ASP.Net. These technology options are certainly understandable when you consider that Klocwork started in the embedded [software](#) space, but such limitations do make K7 less flexible in the enterprise.

Of course, the most important requirement is that a product support your environment. Larger enterprises supporting a range of different development will generally prefer a one-size-fits-all product supporting the widest possible array of languages, platforms and development environments. However, smaller shops--or those that have standardized on a smaller toolset--won't care about additional technologies as long as their particular combination is supported.

[Continue Reading This Story...](#)




WITH NETWORK COMPUTING  
////////IMMERSE YOURSELF IN THIS STORY

**RELATED LINKS**


- ▣ [Analysis: Automated Code Scanners](#)
- ▣ [eEye's Blink Professional 2.5](#)
- ▣ [Affordable IT: Protocol Analyzers](#)
- ▣ [Rolling Review Introduction: Extrusion-Prevention Systems](#)
- ▣ [Review: Reconnex's iGuard 2600](#)

**IMAGES**

[Click image to view image](#)



**NWC REPORTS**



▣ [Review: Automated Code Scanners](#)

Get the full PDF of this article at NWC Reports.

**NWCANALYTICS.COM**

▣ [Host Intrusion Prevention Systems](#)

We examine host IPS in this Network Computing Analytics Tech Report based on an exclusive survey of enterprise users and in-depth lab analysis.

## Nowhere To Hide

Assuming a [scanner](#) supports your technology needs, quality of analysis should be the next biggest decision factor. We ran a general battery of tests in C/C++, [Java](#) and [C#](#) (if supported) to assess the analysis in four major areas:

- » True-positive ratios (correctly detected vulnerabilities)
- » False-negative ratios (missed vulnerabilities)
- » False-positive ratios (incorrectly marked as vulnerable)
- » Quality of contextual info and [vulnerability](#) details

Fortify SCA was the top performer here as well, primarily because of its detailed contextual information. The way SCA presents the entry-point analysis and call paths associated with vulnerabilities, for example, is particularly helpful when trying to validate the vulnerabilities reported by the scanner. SCA rates severity in terms of "hot," "warning" and "info," and provides several useful methods of using the confidence rating for a particular finding. The confidence rating--ranging from zero to five--is listed in the detailed vulnerability information, but you can change the settings to mask vulnerabilities according to a set of predefined confidence levels. These predefined levels are divided into three successively narrower groups: "broad," "medium" and "targeted."

There's quite a [bit](#) of flexibility in how vulnerabilities are viewed, letting you sort and group in the IDE based on almost any vulnerability attribute--severity, detection method, source location and so on.

One major standout feature of SCA is its ability to trace analysis across a wide range of languages and platforms. In our tests, we traced code from ASP.Net into vulnerabilities occurring in TSQL stored procedures. That alone can be a really big win for large, multitiered Web applications.

Overall, we found SCA's false-negative rates manageable, and it caught almost all the vulnerabilities in our test applications. However, finding many of our sample vulnerabilities forced us to review findings at very low confidence levels, where the false-positive rate grew almost unmanageable. In some cases, false positives could be eliminated by properly specifying validation routines, a capability supported by all the scanners, but that can make the scanner significantly less effective because improper use of validation routines may cause

vulnerabilities to be missed. Even after identifying validation routines, we found that many false positives could be eliminated only by manual review.

Ounce has its own impressive analysis set. As is typical, findings are ranked by severity and confidence level, represented by the classification. What makes Ounce unique, however, is that it has a confidence level specifically for validated findings, appropriately named "vulnerability." We found that almost any finding marked at this level was a genuine vulnerability. This really helped some problems [pop](#) out early.

Unfortunately, we also found that the succeeding confidence levels of "Type I," "Type II" and "informational" were significantly less helpful than the vulnerability classification level. We found many true positives ranked at very low confidence levels, with a significantly worse false-positive rate than SCA provided. However, Ounce detected some vulnerabilities completely ignored by other scanners, including NULL DACL assignments and unquoted executable names passed to CreateProcess(). Ounce also detected some [file](#) and pipe races that Fortify failed to identify, though it didn't detect them all accurately.

Klocwork K7's analysis was significantly less detailed than that of rivals. K7 forgoes separate confidence and severity ratings in favor of a single rating addressing both. It uses a numeric value from 1 to 10 (1 being the most severe) with a corresponding text label. The value of separate confidence and severity classification is debatable in automated analysis, so combining them seems reasonable. K7 also includes the Klocwork Architectural Analysis tool for managing large applications. It provides some interesting ways to augment analysis by, for example, identifying large-scale application patterns and segmenting source code and modules, along with findings. Although this could be considered more of a management feature, it's useful for working with the results provided by the analyzer.

In our C/C++ analysis we found that K7 had a less than acceptable false-negative rate--resulting in a large number of missed vulnerabilities. We attempted to address this by configuring the scanner to [display](#) all detected vulnerabilities and performing some additional configuration, but it still missed many important flaws.

In Klocwork's favor, the false positive rate for C/C++ never grew to anywhere near the levels of Ounce or SCA at their lower confidence ratings. We also really liked some of the additional context information K7 provides, such as computed integer ranges and trace summaries for vulnerabilities like cross-site-scripting. Overall, though, we got the feeling that K7 simply wasn't showing us all the findings that it should have.

K7's Java analysis was much more comparable to that from Ounce and SCA, but it produced significantly more false positives at what should have been higher confidence levels. This can be adjusted through configuration and tuning, but we expected a little better out-of-the-box performance, and weren't completely satisfied with the final results.

### Make Me Your Own

All three products provide straightforward means for managing findings, including annotation, classification and suppression of false positives. However, Fortify SCA again impressed us by supporting the most extensive set of management operations from directly within the development environment. Fortify also provides extremely flexible rule-generation options for reducing both false positives and false negatives.

Klocwork K7 offers general finding suppression from inside Visual Studio, along with a more complete set of management features from inside Eclipse. The Klocwork Architectural Analysis tool also provides capabilities for very detailed finding management and grouping. As with the other products in our review, K7 provides a complete set of features for supporting custom rules and filters.

Ounce offers limited management capabilities from inside the development environment; however, the overall management feature set is quite impressive and about even with what K7 and SCA provide. Ounce treats [bug](#) triage as a separate process and supports extensive management through the use of bundles, filters and exclusions. These features allow logical grouping of similar findings and can be used to suppress future false positive results. Ounce also includes a custom rule-generation feature that is comparable to SCA's and supports the same general

level of customization.

### The Human Touch

We also examined how intuitive the product was to use, how well it integrated into development environments and how useful the management interface was. Fortify's SCA led the pack in developer usability, with good integration and a clean, easy-to-use interface. The overall quality of the [interface](#) is particularly noteworthy given the sheer amount of contextual information provided by Fortify's analyzer.

We really liked the way K7 presented information and integrated tightly with Visual Studio and Eclipse. Many shops will appreciate that K7 presents all its findings just like standard [compiler](#) warnings, providing an easy transition for those already familiar with the Visual Studio environment.

Once lagged somewhat on usability due to the general structure of the application and what we consider some awkward portions of its interface. For example, we found that the windows required constant resizing and manipulation to view different results. We also noted that the SmartTrace feature presented much of the same context as that shown by both SCA and K7, but we found the call graph [format](#) less convenient to use while reading code.

All three products provided very capable management interfaces with a range of options. Because of our focus on the developer's perspective, we didn't test these management interfaces extensively. However, they all provide more extensive capabilities for centralized bug reporting and management, report generation, and general metrics and trending.

### General Purpose Testing Tools

IN ADDITION to examining security-specific analyzers, we had an opportunity to work with some general-purpose testing and quality assurance (QA) tools, including Cleanscape [C++](#) lint, Parasoft Jtest and GrammaTech CodeSonar. None provide the range of language support of the tools we tested.

If you're considering (or are using) a tool like these for QA purposes, we strongly recommend incorporating security analysis into your development and testing. While these tools don't provide the same degree of security analysis as the source-code analyzers in our review, they can still be quite helpful in performing software security assessments and for ongoing security testing during development. They also offer general-purpose QA and functionality testing. Depending on your in-house expertise, you might find that a combination of QA testing and manual review provide reasonable coverage, without the need for a security-specific analysis tool.

### Software Security Analysis Techniques

Static source-code analysis is only one possible approach to assessing software security. Spend some time identifying methods that work best with your software, ideally combining a few different approaches.

#### » MANUAL STATIC ANALYSIS

Manual static analysis, or "program comprehension," is the foundation of any security review. Your developers don't need to become masters at this process, but they do need to understand the basics to effectively assess and remediate software vulnerabilities. In *The Art of Software Security Assessment* (Addison-Wesley, 2006), my co-authors and I present a variety of strategies, tactics, and practical examples for performing manual static analysis.

You can also find a good introduction to the topic in Michael Howard's [IEEE](#) article, "[A Process for Performing Secure Code Reviews](#)" ).

#### » BINARY ANALYSIS

Binary static analysis is a technique for performing a detailed assessment of a compiled binary executable. This is appealing because it doesn't require source code, and thus provides a reasonable approach for assessing closed-source software and libraries.

The big disadvantage, however, is that a significant amount of contextual information is lost when analyzing a binary. This missing information complicates parts of the analysis and requires more effort to map vulnerabilities in the binary to their locations in source code. A variety of techniques can be used to improve the effectiveness of binary analysis, including the use of debugging symbols and advanced reverse-engineering tools. Analysis is often performed manually using tools like DataRescue's IDA Pro and is available through security consultancies and as an automated service from providers like Veracode.

## » FUZZ TESTING

Fuzz testing--or fuzzing--is a method for analyzing a running [instance](#) of a program by submitting semi-random data to exposed interfaces. The technique is named for the fuzz utility used to generate random [character](#) streams in the original 1989 [Unix](#) fuzz tests performed at the University of Wisconsin at Madison. Despite its lack of thoroughness, fuzzing has become an extremely popular testing method for a number of reasons. It's relatively simple, can be performed against any live application instance, and the randomness component can be quite effective at detecting vulnerabilities that evade other techniques.

Fuzzers vary in overall quality, with more protocol-aware technologies referred to as intelligent fuzzers. Protocol awareness makes a fuzzer more accurate by incorporating some level of [protocol](#) logic into the fuzzing activity--producing more accurate results in less time. Web applications are particularly well-suited to protocol-aware fuzz testing because they present a consistent, well-known interface that's relatively easy to test for some common vulnerability classes. Although these techniques don't reach the accuracy of source-based analysis, they still provide the speed and simplicity advantages of conventional fuzz testing.

Most Web fuzzers are standalone applications, referred to as Web [application security](#) scanners. However, Compuware's DevPartner SecurityChecker includes a scanner that's integrated into the Visual Studio development environment. The product didn't meet our criteria for source-code analyzers because of limited language support and due to its approach to analysis, but we did perform some basic testing. Although SecurityChecker is limited in its source-code analysis, we found that there are some definite advantages to having the Web application scanner integrated directly into the IDE.

*Justin Schuh leads the application security practice at Neohapsis and has an extensive background in software development and security. He is the co-author of [The Art of Software Security Assessment](#) (Addison-Wesley Professional, 2006) and a frequent poster to the [taossa.com security blog](#).*



[Copyright 2009 United Business Media LLC, All rights reserved.](#)