



## Analysis: Automated Code Scanners

Posted by Justin Schuh on April 13, 2007

Download A Free PDF Of This Article >  
www.informationweekreports.com



Remember when attackers were just out for fame and glory, and [application security](#) was someone else's problem? Big targets like Microsoft and [Oracle](#) drew the fire. All enterprise IT had to do was apply patches regularly and keep a properly configured firewall.

Those days are gone. Cracking corporate networks is no longer a kid's game, it's a lucrative criminal growth industry. The attackers who stole 45.6 million credit- and debit-card numbers from TJX Companies were professional enough to remain undetected for at least 10 months.

Meanwhile, major software vendors, including Microsoft, have improved their security practices, which puts niche and in-house-developed [software](#) and Web applications squarely in the bad guys' sights.

### [Data Privacy Immersion Center](#)

NEWS | REVIEWS | BLOGS |  
FORUMS TUTORIALS |  
STRATEGY | MORE

It seems enterprise IT is finally grasping the liability insecure coding practices represent. Data protection and application-software security were chosen as the most critical issues through 2008 in the 2006 CSI/FBI [Computer Crime and Security Survey](#), above policy and regulatory compliance, and identity theft/data-leakage prevention.

If you think your network's not at risk, consider that most software isn't built for commercial distribution; it's developed in-house or on contract for specific requirements. Purpose-built apps provide the framework for a huge range of business processes, from dynamic Web sites, SOA (service-oriented architecture) and [e-commerce](#) to business process automation and administration. They also provide a target-rich environment for would be attackers.

In response to this escalating threat, major [compliance](#) standards like [HIPAA](#) and PCI [DSS](#) (Payment Card Industry Data Security Standard) are incorporating--or at least implying the necessity of--application security processes.

Of course, where there's a regulation, there's a marketing op. In this case, makers of automated source-code analysis tools are shifting their focus from commercial software vendors to enterprises. They say adopting their tools will let your developers build more secure software and meet the compliance burden.

But are they up to the job?


To find out, we brought three popular static source-code analyzers into our Chicago Neohapsis partner lab: Fortify SCA (Source Code Analysis) 4.0, Klocwork K7.5 and Ounce Labs' Ounce 4.1. We also asked Coverity to send its Prevent analyzer, but the company declined, citing insufficient resources.

Each product has strengths and weaknesses, but any would be a useful addition to a mature security process. And therein lies our most important point: A code scanner, no matter how effective, is only part of the answer. Without adequate developer training and an SDLC (software development lifecycle) that makes security a priority, no tool will protect your network. Sound familiar?

We've long advocated a defense-in-depth strategy that recognizes that there is no perimeter. Security groups have implemented technologies like host intrusion prevention, NAC and database-extrusion prevention that seek to thwart attackers who have infiltrated outer lines of defense before they gain access to sensitive information.

But that's no longer enough. It's time for enterprise IT to partner with in-house and contract developers to make security Job 1. The application development group and the security group all sit under the CIO, so even though security pros typically have had a different world view from developers, who are mainly concerned with providing the functionality requested by the business, political issues can be overcome.

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




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

**RELATED LINKS**


- [Review: 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**



[Analysis: 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.

A Child Can Use It. Really.

Along with most every other IT vendor, makers of application-security scanners are piling on the compliance gravy train, claiming that everyone from QA managers to project leads will find their wares provide valuable insight.

This is marketing spin, not reality. These are tools for developers and software-security pros. Claiming an IT generalist can operate a source-code [scanner](#) is akin to saying a person can use a spell checker on a foreign-language [document](#) when he understands neither the language nor the subject matter. These tools do provide defect-tracking features that are useful at many levels, but all that is contingent on results first being reviewed by a security-knowledgeable developer.

Ultimately, you can't automate your way to secure software--any combination of analysis tools is only as effective as the security knowledge of the user. As Michael Howard, co-author of *Writing Secure Code* (Microsoft Press, 2002) and one of the lead architects of Microsoft's current security posture, put it in his blog: "Not-so-knowledgeable-developer + great tools = marginally more secure code." Producing applications that can withstand attack mandates that developers make fundamental adjustments to the SDLC. We recommend taking advantage of documented processes, including Microsoft's SDL (Security Development Lifecycle), The Open Web Application Security Project's CLASP (Comprehensive Lightweight Application Security Process) and general techniques available at the National Cyber Security Division's "Build Security In" portal.

Analysis technology is advancing rapidly as well, with every vendor constantly trying to leapfrog rivals. That's good for customers, but it makes for a dynamic market. This is one technology that absolutely requires a pilot test to ensure a specific automated source-code analyzer is the right choice for your applications and development

environment. In "Code Scanners Run the Gauntlet" (find it at [nwcreports.com](http://nwcreports.com)), we discuss breadth of languages, platforms and development environments supported.

Think of an analysis tool as a multiplier: In the hands of security-conscious developers, and with appropriate supporting process, a good code analyzer can help your company produce much more secure software. However, if your developers have zero security knowledge or have zero process to support them, you'll see zero benefit.

### Framing The Problem

It's provably impossible to show that any real-world program is free of bugs. And when you consider that security vulnerabilities are really just a particular species of software bug, the deck seems stacked against us.

But hey, when have engineers and practitioners not rushed in where scientists and mathematicians fear to tread? After all, you don't need software to be provably secure, as long as it's effectively secure in practice. That's why security pros have spent decades devising software security techniques, including automated static source-code analyzers.

Static source-code analysis is pretty much what it sounds like--a method for identifying flaws in software by reviewing the application source code. A skilled auditor can analyze [source code](#) manually by reading through it, building a mental model of how the application works, and reasoning through what vulnerabilities could occur. This is by far the most thorough method of software assessment because it incorporates the deepest possible understanding of the internals of the software itself.

The downside of manual static analysis is that it's the slowest form of review and requires a great degree of skill to perform effectively. To put it into perspective, Microsoft estimates that an average developer can review about 1,100 lines of [C++](#) code a day, looking for complex security bugs. Consider how that would scale to even a relatively small application of, say, 50,000 lines. Of course, a skilled auditor or taking a team approach can shrink that window significantly, but that means hiring an expensive security expert or investing serious time and money training your own people.

The expense associated with manual analysis has led to the rise of automated analysis tools. The earliest example is the venerable lint, which appeared as a general quality-checking tool in 1979 as part of the [Unix 7](#) distribution. Modern source analyzers have grown significantly more advanced since then, incorporating a range of advances primarily from the field of [compiler](#) research. These tools can greatly reduce the time required for a software security review when employed by security-knowledgeable developers or professional software security auditors.

### Limitations Of Automated Analysis

Before you can use automated source-code analysis effectively, you need to know what they can and cannot do.

The first limitation involves categorizing potential vulnerabilities to determine proper handling. Consider an SQL injection vulnerability that lets an attacker bypass authentication at a Web site. Would you categorize it as an [SQL injection vulnerability](#) or an [authentication bypass](#)? How about an integer overflow that eventually results in a [stack buffer](#) overflow? Which particular line in the code constitutes the vulnerability?

A human auditor must reason through these kinds of questions, find a logical [thread](#) joining the vulnerabilities, and account for any exceptions or potential confusion. Automated analysis tools cannot reason at that level, so they often identify potential vulnerabilities in unintuitive ways. For example, they may provide confusing classification ratings or insufficient contextual information; mark the same issue multiple times, often as different vulnerabilities; or mark the vulnerable location as deep inside code, when the logical [instance](#) is in an earlier [function](#) call. Because of these categorization difficulties, [output](#) from the tool often requires significant manual analysis to be genuinely meaningful.

Moreover, logical vulnerabilities defy automated analysis. Code scanners cannot detect anything other than basic implementation flaws--and even then, only a subset. They simply lack the capacity to perform meaningful

analysis of application logic and design because these are abstract notions expressing the intent of the developer. They don't map directly to patterns in the program structure. All scanners can do is identify known-vulnerable patterns exposed to user-malleable data, and even on that, they make a lot of compromises to complete analysis in a reasonable amount of time.

As a result, you can't ever expect a tool to perform effectively against errors in permission policy, backdoor credentials, vulnerabilities without known patterns, or logical implementation and design flaws. Unfortunately, these account for more than half of real-world vulnerabilities. And it can be very difficult to determine exactly which vulnerabilities a scanner will miss.

In a perfect world, software vulnerabilities would all fall into strict classifications as design or implementation problems. In that same perfect world, you could rely on the scanner to handle the implementation and concentrate manual efforts on high-level logic and design. Unfortunately, the reality is that vulnerabilities exist on a continuum, from the most abstract design and [architecture](#) bugs to the simplest concrete implementation flaws. Because there is no clear distinction and analyzers vary greatly in what they can detect, you'll need to review significant portions of your code manually for logic or design vulnerabilities. A well-documented design can make this easier, but even the best designs won't account for all implementation logic.

### Lost Cause?

The reality is that any analysis tool is attacking an essentially unsolvable problem. To put it into perspective, consider a simple function that accepts three [16-bit](#) integers as [input](#) and is affected by no other external factors. If an analyzer attempts to handle all possible inputs, it would need to handle  $(2^{16})^3$  different cases--more than 280 trillion test cases! Clearly, the explosion of test cases would become unmanageable in even the simplest of programs.

Because computing all test cases is unrealistic, the analyzer reduces what it needs to test by attempting to determine the boundaries of the input. For the 16-bit integer example, this could be the largest and smallest values allowed for each parameter. The analyzer performs its testing primarily against the boundary cases it identifies.

Clearly then, analysis quality is directly affected by what the analyzer perceives as boundaries, and how it generates its own simplified model of the application. Mistakes in this simplification process result in the analyzer failing to detect real vulnerabilities (false negatives) and identifying vulnerabilities where none exist (false positives). These problems become apparent when analyzing string [processing](#) code, which creates such wide-ranging potential input that the tools have significant difficulty identifying true vulnerabilities.

All three products we tested provide some degree of confidence rating for potential vulnerabilities. They also include capabilities for fine-tuning the analyzer's performance through user-defined rules, false-positive suppression and other product-specific means. Even after extensive tuning, however, you must assess the scanner output and review the code manually.

Finally, high false-positive rates are more than a nuisance--eventually, they'll erode the user's confidence in the tool. When false positive rates are too high, most developers will start to mark any nonobvious finding as a false positive, or just ignore the tool's output entirely. The worst case occurs when developers get fed up, assume all the results are wrong, and alter code to eliminate future alerts (true or false) but fail to address the vulnerabilities.

### Getting To The Source

After analyzing C/C++ and [Java](#) source code in Visual Studio and Eclipse on [Windows](#) using our three source-code analyzers, we discovered a few consistent trends: Some flaws, such as format-string vulnerabilities and double-free bugs in [C](#) code, were identified quite reliably. However, other [bug](#) classes were not handled to anywhere near our expectations. In particular, arithmetic vulnerabilities, such as integer overflows and type conversions, were usually missed or detected only at confidence levels that included an extremely high ratio of false positives. We found this a [bit](#) disconcerting given the growing trend in reports of these vulnerabilities--in fact, integer overflows rose to the No. 2 position in [OS](#) vendor advisories in 2006, just behind buffer overflows,

according Mitre's October Common Weakness Enumeration report ([cwe.mitre.org/documents/vuln-trends.html#overall\\_trends](http://cwe.mitre.org/documents/vuln-trends.html#overall_trends)).

Complex string-handling vulnerabilities often showed the same trend, appearing at low confidence levels, along with a number of other issues common to C/C++ applications. In our Web testing, we even had difficulty detecting a few relatively straightforward SQL injection and cross-site scripting vulnerabilities, though the majority of these vulnerabilities were correctly identified.

Still, all three source-code analyzers performed consistently better than the automated analysis tools we've used in the past, including general-purpose fuzzers and Web application scanners (see "[Software Security Analysis Techniques](#)", for more on other code-testing options). We also found that a bit of tinkering--code restructuring, general cleanup and analyzer tuning--made some vulnerabilities detectable. However, every analyzer missed some severe vulnerabilities and marked significantly more with extremely low confidence ratings.

One important thing to remember is that your results are highly dependent on your environment. We developed our tests to be as general as possible and provide a good representative sample of real-world software. However, even subtle differences can have a significant impact on the analyzer, so the only way to really gauge a tool's performance is to test it on your own code. Fortunately, most vendors support trials, so take one for a test-drive.

FYI: Too little too late: Security testing is done at the programming phase by 5 percent to 10 percent of enterprises, at the testing and predeployment phases by approximately 20 percent of enterprises, and at the operation phase by approximately 70 percent of enterprises, according to Gartner.

FYI: Merge Ahead: By 2009, 40 percent of organizations will use a single vendor that provides both Web application and source-code security scanning features, according to Gartner. And, by 2008, 80 percent of the major SDLC vendors will offer application security scanning tools as part of their [SDLC](#) platform.

FYI: Expensive Bugs: Network downtime caused by security attacks is costing large enterprises more than \$30 million a year, according to a recent study by Infonetics Research. Large companies take the biggest hit from security-related downtime, but results show that small and midsized businesses (SMBs) stand to lose about half a percent of their annual revenue to security-related downtime, which can translate to hundreds of thousands of dollars annually.

### The Compliance Perspective

Since 2000, regulatory compliance actions have targeted companies for data spills linked to application vulnerabilities. Currently, the Federal Trade Commission [Web site](#) lists settlements made with Guidance Software, DSW Inc. and a real estate services company, Nations Holding Co. Could an automated scanning tool have helped these organizations escape compliance liability?

Not on their own, no. Merely remediating validated hits from an automated code scanner doesn't constitute "reasonable" or "appropriate" application security measures. Think of it this way: If you don't own a calculator, buying one may help you do your taxes, but it's a stretch to point to your calculator during an audit. By the same token, if scanning alone fails to find large and obvious classes of vulnerabilities--including the most critical ones--then remediating based on the results of scanning alone falls far short of compliance. For more, see "[Weak Application Security Equals Noncompliance](#)". --*Dave Stampley, Neohapsis general council*

### Microsoft Builds In Security Analysis

IF you're using Visual Studio 2005 Team Edition, you're just a few clicks away from some handy security analysis. The current version includes the PreFast static source-code analyzer, which can be enabled under your project's options or by using the `/analyze` [switch](#) on the command line.

PreFast runs during compilation and generates compiler warnings to inform you of potential C/C++ vulnerabilities. We performed some basic tests, and PreFast successfully detected a number of simple

vulnerabilities without identifying any false positives. IT can make analysis significantly more effective by using SAL (standard annotation language) to eliminate false positives or identify relationships between variables, such as associating a length variable with its buffer.

Visual Studio also includes FxCop, a utility that performs analysis on managed code assemblies to check for conformance with the Microsoft .Net Framework Guidelines. These checks include 26 security-specific tests for safe [.Net](#) assemblies. Many of the issues identified by FxCop are not independent vulnerabilities, but they are risky coding practices that violate specification compliance.

In addition to the FxCop tests, Visual Studio 2005 Team Editions include functionality for automated application and unit testing. Although such testing capabilities are not security-specific, they can be applied to security testing at various levels.

#### NWC Reports: Automated Source-code Security Analyzers

We asked for products that perform automated analysis of source or binary program code for security vulnerabilities. To be qualified, products must support security-specific analysis of Java and C/C++ source code and provide context and guidance for eliminating vulnerabilities identified through automated analysis.

#### PARTICIPATING VENDORS

Fortify Software, Klocwork and Ounce Labs

#### TESTING SCENARIO

We tested all scanners against a set of sample applications infested with previously documented vulnerabilities. The following apps were used:

- » C/C++ command-line utility
- » C/C++ named-pipe client-server utility
- » [J2EE](#) Web application
- » Test battery of C/C++ vulnerabilities, including poor string handling, integer overflows, double frees and [format](#) strings.
- » Test battery of Java Web vulnerabilities, including SQL injection, cross-site scripting, path manipulation/ canonicalization, cross-site request forgeries and access-control violations.

In addition, specific tests for [ASP](#) .Net functionality were performed against Fortify SCA and Ounce Labs Ounce.

#### TESTING CRITERIA

- Analysis Value: Rates how effective analysis is, based on the contextual information provided and the accuracy of results, including true positives, false positives, false negatives and the confidence rating associated with each vulnerability.
- Supported Technologies: Rates utility of the supported technologies in an enterprise according to languages, platforms, IDEs and OSs.
- Price: Per seat perpetual licensing, and other licensing options

- **Usability:** Rates the overall usability of the product in assisting in the identification and remediation of software vulnerabilities.
- **Customizability:** Rates the extent to which the product can be customized to improve the quality of its analysis and to support app-specific requirements.

## RESULTS

Fortify SCA is our Editor's Choice. It provides the widest range of technology support along with some of the most useful analysis capabilities and features. But what impressed us most with SCA is that it served as a useful tool for both automated analysis and manual review. It's important that an analyzer support manual audits properly because a developer will need to interpret and act on findings.

Ounce Labs Ounce 4.1 likewise has a really great analysis lineup, and it caught a few vulnerabilities SCA missed. However, Ounce didn't present the vulnerability context as clearly, and it failed to provide consistently useful context and ratings. This combination made manual validation of vulnerabilities more difficult.

Klocwork K7 is less applicable to enterprise software analysis, but certainly appears well-suited for the embedded space. We really liked certain aspects of the K7 [interface](#) and the way it presented information, such as computed integer ranges. Klocwork also caught a few vulnerabilities missed by SCA, though overall both Ounce and SCA provided consistently better analysis. Don't count Klocwork out yet, though: The latest version of K7 should be available by the time you read this, and its new features should address most of the problems we cited in this review. Major improvements include support for Visual Studio 2005, better vulnerability context and more accurate analysis.

### [Go to the Review](#)

*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](http://taossa.com).*

#### [Application Performance](#)

Improve Scale & Global Reach Today. Read APS Best Practice Whitepaper.  
[www.akamai.com](http://www.akamai.com)

#### [Network Security Test](#)

Scan your network for threats. Free Online Network Security Scan  
[network.security.qualys.com](http://network.security.qualys.com)

#### [Network Security + 60%](#)

Uncover Malware Exposure Learn how.  
[www.malware-control.com/network](http://www.malware-control.com/network)

Ads by Google

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