Software Security Assessment Tools Review

2 March 2009

Jointly funded by:

Assistant Secretary of the Navy
Chief System Engineer
197 Isaac Hull
Washington Navy Yard, DC

and

Naval Ordnance Safety & Security Activity
Box 47, Bldg D-323
3817 Strauss Avenue
Indian Head, MD 20640

Prepared by:

Booz Allen Hamilton
McLean, VA

Table of Contents

Table of Contents (cont'd)

Table of Contents (cont'd)

Table of Contents (cont'd)

Tables and Figures

1. Executive Summary

The Software Security Assessment Tools Review provides detailed information about the software security tools available in the market place. With the goal of introducing security-oriented tools into the life cycle of safety-critical applications, this paper focuses on the types of tools that can be applied in a testing environment that may include software ranging from embedded systems to large-scale enterprise applications. This paper reviews software security assessment tools for possible incorporation into a development laboratory environment. The categories of tools evaluated and detailed in this paper are:

- Static Analysis
- Source Code Fault Injection
- Dynamic Analysis
- Architectural Analysis
- Pedigree Analysis
- Binary Code Analysis
- Disassembler Analysis
- Binary Fault Injection
- Fuzzing
- Malicious Code Detectors
- Bytecode Analysis

For each tool category, the analysis includes information on which stage of the software development life cycle the tool category specifically targets—as well as a discussion of other phases that it may be deployed in. This review also discusses the skills required to operate tools from each category—with the more mature tool categories often having the fewest required skills. For each tool category, the review also discusses the assessed benefits and drawbacks associated with the associated testing tools. Table 1 contains a summary table that includes this information with each of the tool categories. Further analysis of specific tools is located in Appendix A, including a complete list of the tools found in this paper assessed against a significant set of characteristics.

Static source code analysis tools proved to be the most mature of the tool categories reviewed in this paper. These tools have proven to be useful in multiple phases of the software development life cycle: including development, testing, and acquisition. Static analysis tool vendors offer multiple versions of their tools focused either towards developers with little background in security to security code reviewers who are fluent both in security and software development practices. Static analysis tools, like many other tools, cannot aid in identify architectural-level flaws. In addition, these tools require that the source code to the application be available.

Source code fault injection tools provide mechanism through which source code can be *instrumented* to induce the code to follow control paths that would be otherwise difficult to test for. These tools have proven to be useful in multiple phases of the software development life cycle, but show the most promise in the testing phase; they can also serve as an effective aid throughout the development phase. Due to the nature of source code fault injection, these tools require a deep understanding of the software being tested. In return, testers achieve greater code

coverage and a lower false positive rate than other testing methods. However, like static analysis tools, these tools require that the source code to the application be available.

Dynamic analysis tools provide mechanisms to interface with the running application and monitor its behavior. Like source code analysis tools and source code fault injection, this tool category is very mature, but only recently have dynamic analysis tools become focused on security issues. These tools can be used throughout the development life cycle, but have shown to be most useful during the development and testing phases. Dynamic analysis tools often require an understanding of the application's build process and its composition—to best identify false positives. In most environments, the primary drawback associated with dynamic analysis is in the high level of expertise required to discern false positives.

Architectural analysis, while not a tool category per se, is a mature set of processes that organizations can use to identify architectural or design defects within the software. A number of tools are available to aid this analysis, with most requiring an understanding of Unified Modeling Language (UML). Architectural analysis is best applied during the design phase of the development life cycle. The primary drawback associated with architectural analysis is the high level of expertise required—testers must have a thorough understanding of both the software itself and of the architectural analysis process.

Pedigree analysis tools are among the newest entrants into the security testing field. While primarily aimed at maintaining a record of the *pedigree* of the code used to develop software (e.g., the specific license and version of the original open source code base from which the code was adapted), these tools can correlate published security vulnerabilities in open source applications against the code within software that may have been adapted from those projects. These tools can be used in development or later in the life cycle. While pedigree analysis tools can serve to greatly benefit organizations in which open source software is commonly used, they rely on publicly available vulnerability information—meaning custom-developed source code or low-profile open source projects may not benefit from these tools.

Like pedigree analysis, binary code analysis tools are new entrants to the software testing field, with only one primary tool available in this space. Unlike many other tool categories, these tools are available for binary code—including commercial off-the-shelf (COTS) products or legacy applications—without potentially violating licensing agreements. As such, binary analysis aims to be used during the acquisition phase of the software development life cycle with little or no required skills on the part of the user. The primary drawback associated with these tools is the difficulty of extrapolating the program's logic directly from the binary—potentially resulting in false negatives from these tools.

Disassembler analysis, like binary code analysis, allows binary code to be examined for security vulnerabilities. In this process, testers aim to develop an intermediate representation (e.g., decompiled code in a high level language) that can be better analyzed than the binary itself. Like binary analysis tools, these tools can be used during the acquisition phase. Often, they are used during the testing phase to ensure that the compiled code will perform as expected. Due to the large amount of manual analysis that must be performed, disassemble analysis requires a high level of understanding of binary code and how compilers generate binary from source code.

Binary fault injection tools provide mechanisms through which safety- or security-related faults can be sent to the application while it is running. These tools have proven to be useful in multiple phases of the software development life cycle, but show the most promise in the testing phase. Unlike source code fault injection, binary fault injection does not require knowledge of the application's source—as such, the potential for false positives is reduced. Binary fault injection can successfully be performed with little specific training. However, additional analysis may be required to determine the best course of action for mitigating any defects identified by these tools.

Fuzzing tools provide mechanisms through which random inputs can be sent to the application while it is running. These tools have proven to be useful in multiple phases of the software development life cycle, but show the most promise in the testing phase. Unlike other injection technologies, fuzzing tools must often be customized for a specific software application to better identify potential defects (purely random input will often be rejected by the application). As with binary fault injection, additional analysis may be required to determine the best course of action for mitigating any defects identified by these tools.

Malicious code detectors are still being heavily researched within the academic community. The goal of these tools is to make a determination as to whether the binary is inherently malicious. Due to the relative infancy of these tools, they are not yet robust enough to be incorporated into a testing regime. While current tools are still under development, researchers aim to provide fully automated solutions for malicious code detection requiring little or no interaction from the tester.

Bytecode analysis tools are similar to both static source code analysis tools and binary analysis tools, providing benefits associated with each. Because bytecode is similar to a binary representation of the source code, it provides more specific information about the program's execution. However, unlike traditional binary, bytecode still provides some high level semantic information about the program—reducing the effort associated with discerning the software logic. There are two primary types of bytecode analysis tools available: those integrated with static source code analysis (specifically tools that scan Java or .NET source) and those developed for pure bytecode analysis. Pure bytecode analysis tools are commonly used in the testing or acquisition phase of the software life cycle and may require the tester to have a basic understanding of bytecode to improve tool accuracy. The primary drawback of bytecode analysis tools is that they focus only on one specific language's bytecode (e.g., Java or .NET), requiring multiple tools for an organization that develops in more than one language.

This review also discusses a number of supporting technologies that testers can take advantage of when analyzing the security of software. While many of these technologies, such as the Software Assurance Ecosystem, are not yet robust or fully adopted within industry, they may prove useful in the future. Some technologies, such as the Security Content Automation Protocol (SCAP) and related standards, are embraced by government organizations and are being increasingly adopted by vendors.

Summary of Evaluation **Key:** X*- To be most beneficial X+ - In some cases X% - When possible X# - e.g., compilation	Static Analysis Code Scanning	Source Code Fault Injection	Dynamic Analysis	Architectural Analysis	Pedigree Analysis	Binary Code Analysis	Disassembler Analysis	Binary Fault Injection	Fuzzing	Malicious Code Detector	Byte Code Analysis
When to Use											
Requirements				X							
Design				X*							
Implementation	X*	X	X	X	X*	X	X	X	X	X	X*
Testing	X	X*	X*	X	X	X	X	X*	X*	X	X
Production	X	X	X	X	X	X	X	X	X	X	X
Acquisition	X	X	X	X	X	X*	X*	X	X	X*	X
Required Skills (Understanding of…)											
Underlying source code		X									
Underlying development methodology			X#	X					X		
Implementing language	X	X					X%				
Binary						X+	X				
Bytecode											X
Testing methodology		X	X	X			X		X	X	X
Benefits											
Reduces cost over system life	X	X	X	X	X	X	X	X	X	X	X
Educates developers about secure programming	X			X	X						
Rechecks legacy code	X				X	X	X	X	X		X
Automates repetitive and tedious aspects of source code security audits	X				X						
Checks for good programming style	X										X
Increased test coverage		X						X	X		
Increased accuracy		X							X		
No need for source code			X	X		X	X	X	X	X	X
Improved accuracy and coverage		X	X					X	X		
Reduces the amount of testing necessary					X						
No disassembly	X	X	X	X	X	X		X	X		
Guaranteed the analysis is performed on the actual product			X+	X		X	X	X	X	X	X
Drawbacks											
No architectural-level flaws	X		X		X	X	X			X	X
Thorough understanding of the software			X	X				X	X		
Required expertise		X		X			X	X	X		
Requires use of open source software					X						
Lack of tool availability		X				X				X	
Licensing concerns							X				
Reliance on a primary vendor							X				
Additional analysis		X		X			X	X	X		X
Additional preparation		X	X	X					X		
Limited to a single language											X

Table 1: Evaluation Summary of Analyses for Source Code, Executables, and Intermediate Representations

2. Purpose, Scope, and Background

The purpose of this paper is to provide detailed information about software security tools that can also be applied to software safety during the software development lifecycle and system integration. This paper can be used by software experts, as well as the casual user, to select the binary analysis tools, source code analysis tools, and black box testing tools based on the specific characteristics, strengths, and limitations of the tool. Selection of the tool or combination of tools in advance ensures an informed decision is made about the security and safety of the software developed.

This paper evaluates the software security assessment tools for possible incorporation into a testing program. Like most common testing environments, the expected environment in which these tools may be run is a testing workstation running Linux, Solaris or Windows with support for an Integrated Development Environment (IDE). The target testing program for this review is an existing testing program that analyzes developed software for safety-critical concerns. Safety and security are increasingly being recognized as two related properties. For example, if a hacker is able to gain control of a safety-critical system, then it can no longer be considered safe. In contrast, certain security-critical systems may make security decisions that will indirectly affect the safety of users or other entities relying on these systems. Further, increasing adoption of net-centricity and network-enabled systems by DoD exposes many safety-critical systems to threats that had previously been relegated to information sharing systems.

This whitepaper aims to provide an overview of the software security tools available to organizations wishing to implement a security testing program. The scope of analysis in this paper has been limited to information available from individual vendors and other publicly available sources, as well as the authors' experiences while running the tools in different environments and performing analysis of these tools in other contexts. As such, each section will provide a discussion of what stage of the software development life cycle each tool category targets, the generic skills required to operate a certain category of tools, and the benefits and drawbacks associated with each of the tool categories. In addition, this review contains an analysis of the tools currently available on the market in each of these categories in Appendix A as well as an attached, searchable spreadsheet.

3. Problem Space

This section defines the problem space for this whitepaper. Sections include details about the threat environment and properties of interest that will be evaluated by the tools suggested in this whitepaper.

3.1 Threat Environment

This whitepaper assumes that the software components that these tools will be used to analyze are safety-critical software-intensive systems with network connectivity, but are not specifically Web-based. As discussed in Section 2, many safety-critical testing labs are increasingly seeing the need for security testing. As such, the threat environment associated with network-enabled software systems includes malicious external entities as well as malicious insiders. As such, thorough software security testing is necessary to minimize the potential exploitable vulnerabilities that will exist in the software once it has been deployed.

3.2 Properties of Interest in the Software to be Analyzed

Two critical security properties of interest in the software being analyzed are correctness and reliability. Tools that verify these properties include source code analyzers, malicious code detectors, and fuzz testers. These tools verify that the software is operating as expected, and can handle exceptions without impacting the availability of the software or integrity of the data it stores or processes.

Safety is examined from the perspective of the delta between general software “goodness” (e.g., fault tolerance) and survivability. For example, exceptions must be handled and cannot result in shutdown or failure. Tools that assess safety properties include fault injection tools, but exclude tools for gauging general correctness and quality.

3.3 Nature of Software to be Analyzed

This whitepaper assumes that the software to be analyzed is embedded software. Some tools described in this whitepaper will assist the evaluation of embedded software by examining the environment surrounding the software and the ability to penetrate the system to access the embedded software.

3.4 Techniques for Testing

A variety of techniques is available for examining source code and embedded software. This whitepaper examination will include tools with varying levels of interaction, from tool-assisted reviews to near fully automated analysis of binary executables, analysis of source code, and analysis of intermediate representations, including bytecode and the results of disassembly.

3.5 Business Case for Security Testing

In recent years, organizations have found substantial improvements in software security through the use of formal reviews and testing. For example, Microsoft's Security Development Lifecycle (SDL) provides strong emphasis on security testing during software development. Using SDL, Microsoft introduced many security improvements in its operating system (OS) and browser offerings. [1] Similar efforts, such as John Viega's CLASP and SEI's TSP-Secure, have shown observable security improvements in software projects.

By addressing security review results, a significant reduction in security flaws can be achieved—even if these security reviews are performed late in the software lifecycle. As noted by Lavenhar in "Business Case" [2], "there are limited data available that discuss the return on investment of reducing security flaws in source code." Nevertheless, a number of studies have shown benefits through improvements in the software development lifecycle (SDLC) processes [3]. In many environments, software security is often dismissed as a luxury. In other cases, attempts to shorten development schedules or decrease costs lead to reductions in software security testing practices. Security testing is often pushed to the testing or deployment phase of the software development lifecycle. A number of studies have shown that reducing software security and software safety efforts will result in longer development time and more security or safety defects.

One of the earliest studies of the importance of software quality is outlined by Jones in 1991 [4]. "In the 1970s, studies performed by IBM demonstrated that software products with the lowest defect counts also had the shortest development schedules" [2]. These studies identified poor quality as one of the most common reasons for overruns. In 1993, a Software Engineering Institute survey found that more than 60 percent of organizations assessed suffered from inadequate quality assurance [5]. While these studies focused specifically on quality-related defects, security and safety are increasingly being included as an important aspect of quality.

Research has shown that organizations that can prevent or address defects early in the system's lifecycle can achieve significant improvements to the overall cost and schedule of the project.

If an organization can prevent defects or detect and remove them early, it can realize a significant cost and schedule benefit. A number of studies, outlined by Jones [6], have shown that reworking defective requirements, design, and code typically consumes 40 to 50 percent of the total cost of software development [6]. In 1988, Boehm and Papaccio found that every hour an organization spends on defect prevention will reduce repair time from three to ten hours [7]. More recently, through research with the Security Quality Requirements Engineering methodology, Carnegie Melon's Software Engineering Institute has shown that it is possible to optimize security defect mitigation to maximize the benefits outlined by Boehm and Papaccio [8]. Organizations can take advantage of similar methodologies to maximize the quality of the system for their associated budget.

4. Tool Technologies

This section presents the tool technologies reviewed in several categories. The tools described by test technique are defined in Table 2.

Test Technique	Tools		
Static Analysis Code Scanning	Coverity Prevent	Fortify Source Code Analysis Suite	Klocwork K7 Ounce
Source Code Fault Injection	MEFISTO-L	Grid-FIT	
Dynamic Analysis	Daikon	Valgrind	Taintcheck
Architectural Analysis	Enterprise Architect MagicDraw RSA	Together StructureViews	Visustin WithClass
Pedigree Analysis	Black Duck	EULAyzer	Palamida
Binary Code Analysis	Vericode		
Disassembler Analysis	Boomerang Codesurfer	IDA OpenRCE.org	WiSA Simics Hindsight
Binary Fault Injection	Exhaustif	Holodeck	Xception
Fuzzing	GPF	Peach	SPIKE Sulley
Malicious Code Detector	Fakebust	Functional Extraction	
Byte Code Analysis	AspectCheck ASM BCEL FindBugs	FxCop Javassist API JBA JFluid	OPTIMA Bytecode Scanner SofCheck Inspector TL ADT TorqueWrench

Table 2: Primary Tools by Test Technique

Each tool examination will include:

- A description of the technology
- A general assessment of the technology's strengths and shortcomings, including capabilities, expertise required, accuracy and quality of results, support for analysis and interpretation, and integration with other tools
- Examples of specific tools and services-for-hire that leverage the technology (where applicable)
- Where in the system lifecycle these tools are used.

Where services-for-hire are available, the description will include the costs, lead time, and reliability of estimates for those services. The comprehensive matrix of these services is located in Appendix A.

The matrix includes the following criteria against which each tool can be compared:

- Supported languages –Programming languages the tool supports; it is important to note that some tools may be language-agnostic
- Supported platforms where the tool runs – The platforms on which the tool must be run.

- Supported platforms where the target resides – There are cases where the target platform may be different from the tool platform. This separate field will allow for clarification when necessary
- Supported compilers and IDEs –Compilers or development environments supported by the tool. In many cases, the tool will be compiler-agnostic or supported by popular IDEs
- Remote usage – Some tools may not require access to the physical platform on which the target resides
- Finds or checks for (tool category) – Each tool may be a member of one or more of the tool categories outlined in this paper. As such, readers may wish to view only tools from a certain set of categories
- Lifecycle position – While many tools can be used at any point within the software development lifecycle, readers may be interested in identifying the tools available for each specific point in the lifecycle
- Scalability – This field will signify what level of complexity is supported by the tool. However, scalability is a difficult criteria to measure and may not be available for all tools
- Ability to identify comments in code – Tools that can identify comments in code can aid reviewers in both understanding the code itself, as well as locating information that may have been inadvertently left in the comments
- Ability to identify debug code – Tools that can identify debug code can aid reviewers in locating potential security vulnerabilities related to the execution of unintended debug or testing code
- Ability to discover unused code – The ability to identify unused code can aid reviewers in finding sections of the application that may have be inadvertently unreached or portions of the application that may be activated in certain conditions
- Tool uses Common Weakness Enumeration (CWE) definitions – This field identifies whether the tool supports the Common Weaknesses Enumeration
- Frequency of rule base updates – This field signifies how often the tool's database is updated. This is usually a higher frequency for commercial tools than open source tools
- Ability to modify existing rule bases – This field signifies whether the tool supports custom modifications to specific rules
- Ability to modify add rule bases – This field signifies whether the tool supports custom extensions to the rule base, allowing organizations to add their own specific requirements
- Ability to provide mitigation suggestions – This field notes the level of skill required to operate the tool. Tools that provide mitigation suggestions can be operated with less background knowledge than other tools
- Cost – This field identifies how the pricing model of the tool is structured. It may be infeasible to provide explicit cost information for most commercial tools
- Licensing – This field identifies how the tool is licensed. For open source tools, this will specify the license under which the tool is readily available
- Vendor technical support – Organizations may require extra paid support while some tools, particularly open source tools, may offer little or no support
- Vendor services support – This field will signify whether the vendor will provide custom services to an organization

- Required training/expertise – This field will outline what expertise or training is required before a reviewer can use the tool
- Vendor training available – This field will outline what training is offered by the vendor or third parties for reviewers interested in becoming proficient in use of the tool.

4.1 Analysis of Source Code

This section provides an overview of information on the techniques available for analyzing source code.

4.1.1 Static Analysis Code Scanning

As identified by Michael and Lavenhar in *Source Code Analysis Tools – Overview* [9], a static source code security analyzer is a tool that aids analysts in locating security-related issues in software. According to the authors, "The impetus for security analyzers originally came from the realization that many software vulnerabilities are found in reusable library functions; thus, programs could be scanned to check whether they contain any calls to those functions. This process is similar to opening the source code in an editor and searching for the name of vulnerable functions like strcpy() and stat()" [9].

Modern security analyzers are more sophisticated, with the ability to use data- and control-flow analysis to locate subtler bugs and reduce false positives. While security analyzers are used to make human analysts more efficient by automating certain mechanical tasks in the review process, these tools are still not capable of replacing a human analyst [9]. Nevertheless, one important aspect of modern security analyzers is that many act as a compiler or rely on the output generated by compiler front-ends[1] to better analyze the structure and flow of the program. It is important to note that modern compilers, to an extent, act as security analyzers—they offer protection against buffer overflows and when all warnings are enabled, many security-related warnings can be presented to the developer at compile time. Moving some security analysis to the compiler has shown to be an effective technique for mitigating some of the most egregious security vulnerabilities (e.g., overflow errors and the use of unsafe library functions).

4.1.1.1 When to Use Static Analysis Tools

Static analysis tools are frequently used during source-code audits and walkthroughs, with the primary goal of increasing security analysts' efficiency [9]. In addition, most static analysis tools can be integrated within developers' IDEs so they can be used during development, increasing the potential of detecting and mitigating vulnerabilities earlier in the lifecycle of the application. Even though static analysis tools can be used throughout the development lifecycle, they should be employed as early as possible. Problems found earlier are usually easier and less expensive to fix [9].

[1] Modern compilers are composed of two parts. The front-end processes the source code and generates parse trees, which are then used by the back-end to generate the binary code.

The limitations of static analysis tools must also be considered. Static analysis tools are not able to detect most design- or architecture-level flaws; they are meant to work on source code. Extrapolating design- or architectural-level details from the source code is a hard problem that researchers are still working on. Similarly, static analysis tool scanners are not currently well suited for finding integration bugs that may occur between application components.

Some static analysis tools are designed to enforce coding-style best practices and act as coding-style checkers. The relationship between security-oriented static analysis tools and coding-style checkers is increasingly being blurred. Nevertheless, to be effective in this role, these tools must be used early in the development process. It is often prohibitively expensive to bring the software into compliance with analyzer-enforced guidelines [9].

Those analyzers that try to deduce software behavior usually emphasize technology that reduces false alarms (e.g., apparent vulnerabilities that actually turn out to be unexploitable). Reducing false alarms also reduces the amount of work required to fix issues an analyzer finds. While it is not a good practice to postpone security analysis, these analyzers might be useful in situations where security analysis cannot begin during the early development stages, such as in cases where legacy code has to be audited.

One of the primary features of modern static analysis tools is the use of technology that reduces false positives[2] [9]. Because one of the primary roles of the security analyst is to verify the tool's findings, reducing false positives reduces the amount of work the analyst has to perform.

Because they work directly against the source code, static analysis tools can be used at any point in the development lifecycle where source code artifacts are available. It is best to begin performing security analysis as soon as code is being generated (i.e., by integrating the static analysis tool with the developers' IDEs), but in situations where an organization cannot deploy the tool until later in the software's life cycle, the results from a static analysis tool can still be highly beneficial to understanding an organization's security posture.

4.1.1.2 Required Skills

Most static analysis tools are targeted at users with varying degrees of skills, from developers to security analysts. For developers, these tools provide a description of the vulnerability and tips and techniques for mitigating the vulnerability. Static analysis tools are more powerful in the hands of experienced security professionals who have a strong background in the application's source language. With an understanding of the development language and basic information assurance concepts, the analyst can better determine whether a specific finding is a false positive or a mitigation strategy that fits better with the design goals of the application as a whole.

4.1.1.3 Potential Benefits

Static analysis tools have the following potential benefits:

[2] False positives are vulnerabilities identified by the analysis tool that are not exploitable.

- Reducing costs over the system lifetime – Security vulnerabilities identified early in the lifecycle are cheaper to fix
- Educating developers about secure programming – The list of insecure programming practices is long and continues to grow. This makes it difficult for developers to ensure full compliance with secure programming guidelines, especially since most developers are not trained in secure programming. A good static analysis tool not only finds problems, but also explains what is wrong and how to fix it. This provides developers with hands-on feedback on how to improve their own programming practices
- Rechecking legacy code – Even if the legacy code was developed with security in mind, it is inevitable that new classes of vulnerabilities will have come to light since the code was developed. Static analysis tools can help identify these issues
- Automating repetitive and tedious aspects of source code security audits – Static analysis tools free up human security analysts to identify more difficult problems, such as architectural or design flaws, that have security implications
- Checking for good programming style – These tools can often check coding style from a security standpoint. Keep in mind that the tool will check for its own idea of what constitutes "good style" unless it is customized. Customization can be time consuming and requires a certain amount of expertise.

4.1.1.4 Drawbacks

Static analysis tools are not designed to find complex architecture- or design-level flaws, and they are not well suited for finding integration bugs. These tools are also somewhat limited when it comes to analyzing large systems, such as those consisting of many executable components or heterogeneous layers. It should be emphasized that, like human auditors, static analysis tools will not find all of the vulnerabilities in a software system. A static analysis tool will only be able to find a certain subset of the vulnerabilities in a given piece of software, and that subset will not change unless the scanner is reconfigured or updated with a newer version. Static analysis tools look for a fixed set of patterns or rules in the code. Although more advanced tools allow new rules to be added over time, if a rule has not yet been written to find a particular problem, the tool will never find that problem. Therefore, it is important not to rely on static analysis tools as the sole means of securing software source code.

4.1.1.5 Static Analysis Tools

A list of the analyzed static analysis tools are below.

Product: Coverity Prevent

Description: Finds quality issues and security vulnerabilities using static source code analysis in C and C++ code. It is a pure analyzer with a simple interface. This product has no management console. The only way to see what has changed between runs of the analyzer is to run diff, a programming utility from Unix that identifies what has changed in a source file. Dashboards or other displays of project status are nonexistent. Further details are described in Table 3.

Supported Languages	Detailed Description		Supported Platforms
Supported C/C++ Compilers Arm CC G++ GCC Green Hills compiler HP-UX compiler IAR compiler Intel compiler for C/C++ Intel Microsignal Architecture compiler Java JDK (1.4 and higher) Metrowerks CodeWarrior MS Visual Studio PICC compiler Sun CC TI Code Composer C compiler Wind River Diab compiler	**Checks include:** API usage errors Buffer overflow Cross-site scripting Dangling stack references Denial of service File corruption Flawed branch logic Format string vulnerabilities Improper bounds checking Incorrect allocation sizes Insecure access control Integer overflows Logic errors	 Memory corruption Memory leaks Non-null terminated strings Null pointer dereferencing Out-of-bounds array access Privilege escalations SQL injection Stack overflow Stack smashing Stack string overruns System resource leaks Use of freed resources Use of uninitialized data	FreeBSD HP-UX Linux Mac OS X NetBSD Solaris Sparc Solaris X86 Windows

Table 3: Coverity Prevent Additional Features

Product: Fortify Source Code Analysis (SCA) Suite

Description: Analysis is conducted at a semantic, rather than syntactical, level. This means that the product understands what the code is doing. For example, the product can map out data flows and recognize that untested, user-entered data has been passed to a routine. The routine may be correct in its functioning but unaware that the data passed to it has been corrupted in a way that unhinges the application. Because the Fortify engine understands the code, it can monitor execution and data flows through multiple modules and identify the points where unsafe data is touched without first being verified. Few solutions can find intermodule security problems of this kind.

Fortify generates a large XML file containing data on all identified vulnerabilities. This file is then analyzed by the Workbench, which displays the information in a user-friendly format. Unless programmers are up to date on the nature of specific coding vulnerabilities, they are likely to find new items flagged by Fortify. The product catches not only buffer over-runs and opportunities for SQL injection, but also more esoteric issues.

Because the number of generated warnings can be large, the Audit Workbench automatically assigns severity ratings and enables the creation of filters so that only items of interest are displayed. The display not only lists the vulnerabilities and explanations, it also takes developers directly to the specific line of code.

Fortify SCA results can integrate with Fortify software's centralized, web-based reporting and control console to make findings and metrics, including trends within projects, comparisons between groups, and more, available to key stakeholders. Further details are described in Table 4.

Supported Languages	Detailed Description		Supported Platforms
Supports mixed language analysis of ASP.NET, C/C++, C#, Java, JSP, PL/SQL,T-SQL, VB.NET, XML, and other .NET languages. **Supported IDEs** Microsoft Visual Studio IBM Rational Application Developer Eclipse	Checks over 118 vulnerability categories including: Buffer overflow Command injection Cross-site scripting Denial of service Format string Integer overflow Log forging	 Password management Path manipulation Privacy violation Race condition Session fixation SQL injection System information leak Unreleased resources	Windows, Solaris, Linux, HP-UX, AIX, and Mac OS X

Table 4: Fortify Source Code Analysis Suite Additional Features

Product: Klocwork

Description: This tool seamlessly integrates into today's most popular development environments. Klocwork's patented static source code analysis technology extends management insight, auditor analysis, and developer assistance across the following critical development challenges:

- Automatically detects operation-affecting defects early in the process
- Finds security vulnerabilities in software and improves overall application security
- Understands large code bases and simplifies their structure
- Measures and tracks key quality indicators throughout the release cycle
- Allows for customization of the analysis to suit an organization's quality and security priorities.

K7 can perform analysis based on Java source code and bytecodes, the latter being Java's form of an executable file. If the bytecodes contain debug information, K7 can trace defects back to specific lines of code. If not, the tool can identify that a certain type of bug has been found. This option enables sites that rely on third-party Java components to screen them for possible defects before use and to identify the type of defect to the vendor.

A separate utility presents a detailed pictorial analysis of the complex relationships between files and functions. K7 can identify relationships that would indicate bugs, such as a library of functions making calls to an application. K7 also has extensive reporting capabilities. The management console can generate a PDF (filters enable managers to include or exclude a wide variety of data), exportable text, or XML files. Further details are described in Table 5.

Supported Languages	Detailed Description	Supported Platforms
C/C++ Java **Supported Compilers** GNU GCC/G++ ARM Microsoft Visual C++ Green Hills Wind River Diab Sun Forte MetroWerks Metaware Hitachi h38 TI tms470 Sun Java Compiler **Supported IDEs** Eclipse 3.0, 3.1, 3.2 Microsoft Visual Studio 6, 2002, 2003 IBM Rational Application Developer 6.0 Wind River Workbench 2.3, 2.4, 2.5 QNX Momentics 6.3 (SP2)	**C/C++ Vulnerability Categories** Access problems Buffer overflow DNS spoofing Ignored return values Injection flaws Insecure storage Unvalidated user input **Java Vulnerability Categories** Denial of service Injection flaws (e.g. SQL injection) Unvalidated input Mobile code security Broken session management Cross-site scripting Improper errors handling Broken access control	Sun Solaris 8-10, Red Hat Linux 9.0, Red Hat Enterprise Linux 3.0 - 4.0, SUSE Linux 9.3, Windows XP Professional, Windows 2000 Professional, Windows 2000 Server, and Windows Server 2003

Table 5: Klocwork Additional Features

Product: Ounce

Description: Ounce automatically analyzes source code through the use of a language processor that parses the application to create a Common Intermediate Security Language (CISL). The CISL captures multi-dimensional information about each call site, allowing Ounce to refine vulnerability data through three different levels of analysis. Ounce determines vulnerabilities by tracking the flow of data through an application, using cross-module, cross-language, semantic, and data flow analysis to understand the complex interrelationships between individual calls, modules, data elements, and processes.

Ounce separates real vulnerabilities from potential ones, allowing security analysts, quality assurance (QA) teams, and developers to instantly click to confirmed vulnerabilities for focused remediation efforts. Ounce also sorts results by severity (high, medium, and low) and by type (buffer overflow, race condition, privilege escalation, etc.), and Ounce's Security Knowledgebase offers suggestions to the developer for correcting the vulnerability or exception. Ounce allows the developer to make the choice to correct or modify the code on a case-by-case basis because the developer typically understands more about the desired behavior of the application. Customers may tailor the Security Knowledgebase to specific security and policy standards and apply those standards consistently across the enterprise. Further details are described in Table 6.

Supported Languages	Detailed Description	Supported Platforms
C/C++, Java, JSP, ASP.NET, VB.NET, and C# **Supported IDEs** Microsoft Visual Studio IBM Rational Application Developer Eclipse	**Vulnerability Categories** Buffer overflows Privilege escalation Race conditions in C and C++ Input validation errors SQL injection vulnerabilities Cross-site scripting errors Basic design flaws (such as proper implementation of access control) Basic policy violations (e.g., is cryptography in use and is it strong enough?)	Windows, Solaris, Linux, and AIX

Table 6: Ounce Additional Features

4.1.2 Source Code Fault Injection

According to the Information Assurance Technical Analysis Center (IATAC) State of the Art Report on Software Security Assurance, fault injection is a form of dynamic analysis in which the source code is "instrumented" by inserting changes, then compiling and executing the instrumented code to observe the changes in state and behavior that emerge when the instrumented portions of code are executed [10]. This is separate from the form of fault injection discussed in Section 4.2.3, where the faults are injected through the application's inputs.

In fault injection, the goal is to analyze the effects of a single fault within the software through the application as a whole. Using this information, the tester can extrapolate the impact of a particular fault on the software as a whole, perhaps leading to exploitable vulnerabilities.

Fault injection has a large following in the software safety field, particularly in environments where the software must continue operating safely regardless of environmental or internal effects. In some cases (e.g., embedded devices), it may not be practical to alter the software to handle any new defects, leading to increased testing requirements. Fault injection has emerged as a common security testing technique. The ability to inject potential changes into the source code itself allows testers to ensure that all possible code paths are tested. In contrast, complicated interactions that lead to a specific code path may be difficult to implement from outside of the system itself.

4.1.2.1 When to Use Source Code Fault Injection

Source code fault injection tools can be used at any point in the development lifecycle where source code artifacts are available. Unlike many forms of source code analysis, the full benefits of source code fault injection may not be fully realized until a substantial portion of the software application is available for integration or unit testing because one of the primary aspects of source code fault injection is in fault propagation analysis. While identifying aspects of the software-intensive system that may be susceptible to faults early on will aid in mitigating these defects, it is likely that the most complex—and potentially the most hazardous—defects will not surface until later in the software development lifecycle. Nevertheless, as with any software analysis tool, it is important to deploy source code fault injection as early in the software development lifecycle as possible.

4.1.2.2 Required Skills

Many fault injection tools are targeted at users with a strong understanding of the fault injection process and the code base against which testing will be performed. Some commercial vendors (e.g., Security Innovation's Holodeck) offer tools that can be used with less experience in this field of testing. Many of the tools discussed in Section 4.1.2.5 are academic in nature, requiring more experience and understanding to use the tools and process their results.

4.1.2.3 Benefits

The benefits of fault injection tools include:

- Increased test coverage – By injecting faults directly into the source code, testers can ensure increased test coverage. This is in contrast to traditional testing approaches where test coverage is determined by providing the appropriate set of inputs to the application.
- Increased accuracy – By specifically modifying the source code to inject faults, testers have greater assurance that a specific condition will lead to a potentially executable or unsafe action within the application. Accuracy can be further increased by devising traditional testing methods that will replicate the injected fault.

4.1.2.4 Drawbacks

One of the primary drawbacks associated with software fault injection tools is the added requirement that the reviewer have a thorough understanding of the software to be tested. This is important to ensure that the faults injected into the software make logical sense. For example, injecting faults into the application that could not occur during a real operating scenario may increase the false-positive rates associated with software fault injection. Ideally, a skilled tester could ensure that there are no false positives in results produced by software fault injection.

A further concern when using software fault injection tools is that they require additional manual labor. Identifying which aspects of the software should be modified for testing requires substantial analysis of the software architecture and data structures in use. As such, the test team should either be fully briefed by the software developers and architects or be involved in software fault injection testing throughout the development of the application.

4.1.2.5 Specific Tools

A number of fault injection tools are available on the market. However, many COTS tools that support source code fault injection more readily identify themselves as binary fault injection tools. Those tools are found in Section 4.2.3.5. The following tools are examples of pure source code fault injection tools:

- **MEFISTO-L** – The Multi-Level Error/Fault Injection in Simulation Tool (MEFISTO) is a source-code fault injection tool for the VHDL hardware description language. Developed in the mid 1990s, MEFISTO is one of the most commonly referenced tools for performing source-based fault injection. *http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?isnumber=7613&arnumber=315656&count=48&index=40*

- **Grid-FIT –** An implementation of the Fault Injection Technology (FIT) framework that performs network-level fault injection on deployments of the Globus Grid and other web services-based systems. *http://www.allhands.org.uk/2006/proceedings/papers/607.pdf*

4.1.3 Dynamic Analysis

According to the IATAC State of the Art Report on Software Security Assurance, dynamic analysis occurs when "the compiled executable is run and fed a set of sample inputs while the reviewer monitors and analyzes the data (variables) the program produces as a result [10]. In *The Concept of Dynamic Analysis* [11], Ball describes two types of analysis:

- Coverage concept analysis produces dynamic control flow invariants for a set of executions, which can then be compared with statically-derived invariants to identify desirable changes to the test suite that will enable it to produce better test results
- Frequency spectrum analysis counts the number of executions of each path through each function during a single run of the program.

Reviewers can then perform a third analysis looking for specific patterns in the program's execution, such as uncaught exceptions, assert failures, dynamic memory errors, and security problems [10].

The most prominent form of dynamic analysis for security testing is in the web application vulnerability assessment arena. Tools from this arena can operate on a black-box implementation of an application and determine to some extent the security posture of the application itself. This is due, in large part, to the relatively simple communications channels between web applications and users (HTTP GET and HTTP POST), allowing these tools to easily identify inputs and outputs associated with the program.

Dynamic analysis tools have been used within the safety and quality community for some time. One of the earliest examples of a dynamic analysis tool is the IBM OLIVER toolkit. OLIVER provided developers with insight into the operation of a running program (similar to modern debuggers) while providing facilities to ensure that certain program errors are prevented during runtime [12].

While modern debuggers (e.g., GDB and the Microsoft .NET debugger) provide some dynamic analysis capability, they are specifically targeted to functional testing. While it is possible to use these tools to augment security or safety testing (many of the tools discussed in other sections rely on the information generated by debugging frameworks), they are insufficient for performing security or safety-related dynamic analysis. Nevertheless, a number of tools are available both commercially and through open source to aid in dynamic analysis.

4.1.3.1 When to Use Dynamic Analysis

The majority of dynamic analysis tools are designed to be used during the development phase of the software lifecycle. Many of these tools are aimed at identifying traditionally difficult-to-test defects. In some cases, these tools focus on defects associated with memory errors or race conditions which may manifest themselves in unexpected ways later in the software lifecycle.

4.1.3.2 Required Skills

Because the way different dynamic analysis tools are run varies, the skills required to use each tool also varies. A number of dynamic analysis tools behave as simple, stand-alone testing tools which are run against an existing application. Other dynamic analysis tools must be linked with the target application, requiring the tester or developer to modify the software's compilation procedures. Many of the dynamic analysis tools aimed at finding memory errors require recompilation and a full understanding of the underlying application.

The Debian OpenSSL libraries offer an example of the effects of misapplying the results of dynamic analysis. In September 2006, changes were made to the Debian version of the OpenSSL libraries to address the results of output from the tool Valgrind, resulting in the introduction of a security vulnerability that was identified in 2008. Because the results of Valgrind were misapplied, many servers were deployed with insecurely generated secure socket layer (SSL) keys [13].

4.1.3.3 Benefits

The benefits of dynamic analysis tools include:

- No need for source code – Some dynamic analysis tools do not require access to the source code, allowing COTS applications to be analyzed
- Improved accuracy and coverage – By integrating with the running binary application, many dynamic analysis tools can improve accuracy and coverage over traditional testing techniques, including source code analysis.

4.1.3.4 Drawbacks

The primary drawback associated with dynamic analysis tools is in the level of expertise required to use them. As the Debian OpenSSL incident shows, users need both an understanding of the dynamic analysis tool and a full understanding of the target code base to adequately address and mitigate the findings from dynamic analysis. Similarly, the dynamic analysis tools identified in this effort require access to the source code.

4.1.3.5 Tools

A number of dynamic analysis tools are available on the market. The following tools are the most commonly used and robust security-oriented dynamic analysis tools identified during this review:

- **Daikon** – While not a security- or safety-specific dynamic analysis tool, Daikon shows the potential of dynamic analysis. Daikon examines running software to provide information about likely invariants, which can be important in understanding the underlying operations or structure of the software itself. *http://groups.csail.mit.edu/pag/daikon/*
- **Valgrind** – Valgrind automates testing for memory-related errors, including memory leaks. Since its debut, Valgrind has evolved into a more general dynamic analysis and profiling system. Valgrind offers a suite of tools that can be used to gain a better understanding of the software, including call graph analysis (callgrind), thread profiling (helgrind), heap profiling, and cache analysis. While none of these tools are inherently security- or safety-specific, Valgrind is highly customizable. *http://valgrind.org/*
- **Taintcheck** – Based on the Valgrind framework, Taintcheck uses dynamic analysis to identify security defects within COTS software. *http://www.stanford.edu/~stinson/paper_notes/tainting/taintcheck.txt*

4.1.4 Architectural Analysis

Paco Hope, Steven Lavenhar, and Gunnar Peterson have defined architectural risk assessment as a process for identifying flaws in software architecture and determining risks to information assets that result from those flaws. Specifically, "The process of architectural risk assessment finds flaws in software that expose information assets to risk, prioritizes the risks based on their estimated impact to the business, develops mitigations for those risks, and reassesses the risk to determine the efficacy of the mitigations that have been developed and implemented. This section examines the architectural risk analysis of software threats and vulnerabilities and assesses their impacts on assets. This section also describes the process of risk analysis, which is broken down into sections on asset identification, risk analysis, risk mitigation, and risk management and measurement" [14].

Figure 1 illustrates the architectural risk analysis process.

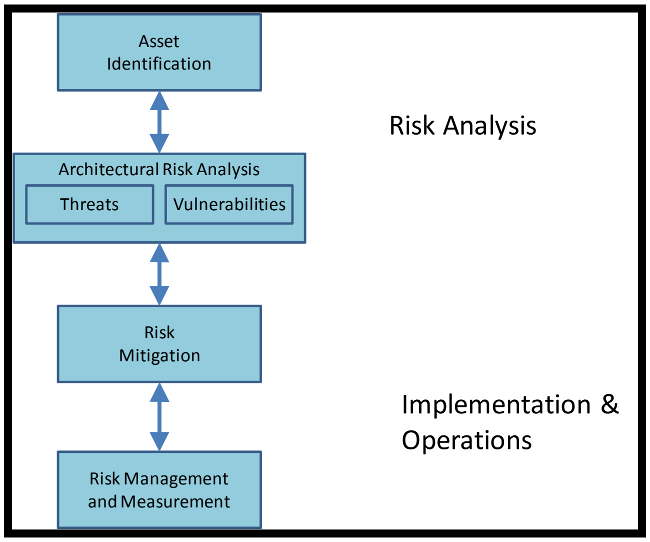
Asset Identification
Risk Analysis
Architectural Risk Analysis
Threats
Vulnerabilities
Risk Mitigation
Implementation & Operations
Risk Management and Measurement

Figure 1: Process View of Risk Analysis and Risk Management Areas [14]

Architectural risk assessment is composed of point-in-time and ongoing processes [14]. As software evolves, code, functionality or entire components may be added or removed, resulting in changes to the architecture itself. Similarly, the body of known attack patterns is always growing. As such, Hope et al. argue that ambiguity analysis is always necessary, though over time it can focus primarily on new requirements or new functionality. Nevertheless, it is important to periodically review the entire system again, taking into account the architectural-level changes that may have occurred since the last risk assessment. Even if the architecture does not change, as upgrades or patches are applied to existing components, existing problems will be addressed but new defects will likely be introduced. The following subsections will highlight the risk analysis process.

4.1.4.1 Asset Identification

The first step in any risk analysis process begins by identifying the assets that must be protected. Assets can be identified as anything of value to the organization. Successful risk analysis depends on the accurate identification of components of the software and any related systems. To successfully identify assets, the reviewer should look beyond the software development team and include the views of the organization or managers deploying the software. This provides insight into the goals and constraints of the software, as well as an understanding of the impact of software failures.

Information assets may take the form of physical hardware, the logical databases on which application information is stored, audit records created by the software, user credentials, or other information related to the software's goals. Beyond identifying each asset, the reviewer should identify which criteria are important for each asset. For example, the integrity of sensor data provided by an embedded system may be more important than the availability of that data. Successful identification of assets can be achieved through a series of interviews with stakeholders.

With knowledge of the specific assets within the system, it should be possible to identify which software modules or components manipulate each asset. Analysis should spiral outward from an asset to determine which software elements read, write, modify, or monitor that information. All information assets should be compiled in a list to be coordinated with risk analysis.

To aid with the understanding of requirements for specific assets, organizations can take advantage of the concept of asset classes defined in National Institute of Standards and Technology (NIST) Special Publication 80-60, *Guide to Mapping Types of Information and Information Systems to Security Categories*, and the Federal Information Processing Standards Publication 199, *Security Categorization of Federal Information and Information Systems*. The asset class, along with exposure and the combination of threat and vulnerability, defines the overall impact to the organization. The impact is then combined with probability to complete a well-formed risk statement.

4.1.4.2 Risk Analysis

Once assets have been appropriately identified, the reviewer can begin assessing the architectural risks for a software system. As part of this process, the boundaries of the software system should be identified, as well as the resources, integration points, and information that constitute the system. Once the boundaries are defined, many artifacts are required or desired for review. These include, but are not limited to, the artifacts in Table 7.

Software business case	Security architecture documents
Functional and non-functional requirements	Identity services and management architecture documents
Enterprise architecture requirements	Quality assurance plan
Use case documents	Test plan
Misuse and abuse case documents	Risk management plan
Software architecture documents describing logical, physical, and process views	Software acceptance plan
Data architecture documents	Problem resolution plan
Detailed design documents, such as UML diagrams, that show behavioral and structural aspects of the system	Risk list
Software development plan	Issues list
Transactions	Project metrics
	Programming guidelines
	Configuration and change management plan
	Project management plan
	Disaster recovery plan
	System logs
	Operational guides

Table 7: Artifacts available for Risk Analysis [14]

Reviewers may also take advantage of system-level documents when performing a risk analysis. For example, the artifacts in Table 8 provide lower-level information about the software system, but are more likely to be readily available than some of the documents described in Table 8.

System documentation and data criticality (e.g., the system's value or importance to the organization)	Information storage protection that safeguards system and data availability, integrity, and confidentiality
Documentation of the system and data sensitivity	Flow of information pertaining to the software (e.g., system interfaces and system input and output flowchart)
System security policies governing the software (organizational policies, federal requirements, laws, and industry practices)	Technical controls used for the software (e.g., built-in or add-on security products that support identification and authentication, discretionary or mandatory access control, audit, residual information protection and encryption methods)
Management controls used for the software (e.g., rules of behavior and security planning)	

Table 8: System-Level Artifacts [14]

The goal of the these activities is to produce one or more documents that depict the vital relationships between critical parts of the system. It is often not possible to model and depict all interrelationships. Using information gathered through asset identification and from security best practices, the diagrams and documents gradually take shape.

Before the risk assessment commences, reviewers should research and clearly understand each element of the assessment. The following information covers best practices and further defines each element in a well-formed risk assessment. Use the following list to help collect material to be used as inputs into the risk assessment process:

- New business drivers – Understand the organization's priorities, including any organizational changes that have occurred, such as mergers and acquisitions, that may have changed the technology base
- Previous risk assessments – If available, review previously performed risk assessments. The risk assessment team may have to reconcile the new assessment against previous work
- Audits – Review any audit reports that are relevant to the risk assessment. Audit results should be addressed in the assessment
- Security incidents – Use past incidents to identify key assets, understand the value of assets, identify prevalent vulnerabilities, and highlight control deficiencies
- Industry events – Identify new trends in the organization and external influences. These include government regulations and laws and industry best practices that may significantly affect the organization's risk posture
- Bulletins – Review known security issues that are identified by organization such as U.S. Computer Emergency Response Team (CERT), the SANS Institute, and applicable software vendors
- Information security guidance – Industry standards can be leveraged to improve or help justify the risk assessment process or identify new control strategies. International standards are another key input.

This guidance incorporates concepts from many standards, including the NIST Special Publication series and the International Organization for Standardization . Careful evaluation and application of standards allows users to share the work of other professionals and provide additional credibility with organization stakeholders. It may be helpful to reference standards during risk discussions to ensure the assessment covers all applicable areas of information security.

4.1.4.3 Threat Analysis

Prior to performing a full risk analysis of the system, it makes sense to identify the threats that face the system. Threats are agents that violate the protection of information assets and site security policy. Threat analysis identifies for a specific architecture, functionality, and configuration. Threat analysis may assume a given level of access and skill level that the attacker may possess. Threats may be mapped to vulnerabilities to understand how the system may be exploited. A mitigation plan is composed of countermeasures that are considered effective against the identified vulnerabilities that the threats exploit.

Attackers who are not technologically sophisticated are increasingly performing attacks on systems without really understanding what it is they are exploiting because the weakness was discovered by someone else. These individuals are not looking to target specific information or a specific company, but rather to use knowledge of a vulnerability to scan the entire Internet for systems that possess that vulnerability. Table 9, developed by NIST, summarizes potential threat sources.

Threat Source	Motivation	Threat Actions
Cracker	Challenge Ego Rebellion	System profiling Social engineering System intrusion and break-ins Unauthorized system access
Computer criminal	Destruction of information Illegal information disclosure Monetary gain Unauthorized data alteration	Computer crime (e.g., cyber stalking) Fraudulent act (e.g., replay, impersonation, and interception) Information bribery Spoofing System intrusion Botnets Malware, trojans, viruses, worms, and spyware Spam Phishing
Terrorist	Blackmail Destruction Exploitation Revenge Monetary gain Political gain	Bomb Information warfare System attack (e.g., distributed denial of service) System penetration System tampering
Industrial espionage	Competitive advantage Economic espionage Blackmail	Economic exploitation Information theft Intrusion on personal privacy Social engineering System penetration Unauthorized system access (access to classified, proprietary, or technology-related information)
Insiders (poorly trained, disgruntled, malicious, negligent, dishonest, or terminated employees)	Curiosity Ego Intelligence Monetary gain Revenge Unintentional errors and omissions (e.g., data entry and programming errors) Wanting to help the company (victims of social engineering) Lack of procedures or training	Assault on an employee Blackmail Browsing of proprietary information Computer abuse Fraud and theft Information bribery Input of falsified, corrupted data Interception Malicious code (e.g., viruses, logic bombs, and trojans) Sale of personal information System bugs/intrusion/sabotage Unauthorized system access

Table 9: NIST Threat Sources

Further complicating the prevention of specific attacks is that an attacker's intent is not always clear. Both internal and external threat sources may exist, and an attack taxonomy should differentiate between attacks that require insider access to a system and attacks initiated by external sources. Internal attacks may be executed by threat actors such as disgruntled employees and contractors. It is important to note that non-malicious use by threat actors may result in system vulnerabilities being exploited. Internal threat actors can act on their own or under the direction of an external threat source (e.g., an employee may install a screensaver that contains a trojan). Internal threat agents currently account for the majority of intentional attacks against government and commercial enterprises.

There are three main types of external threats:

- Structured external threats are generated by a state-sponsored entity, such as a foreign intelligence service. The resources supporting the structured external threat are usually high and sophisticated
- Transnational threats are generated by organized non-state entities, such as drug cartels, crime syndicates, and terrorist organizations. Such threats generally do not have as many resources as the structured threats (although some of the larger transnational threat organizations may have more resources than smaller structured threat organizations). The nature of the transnational external threat makes it more difficult to trace and provide a response.
- Unstructured external threats are usually generated by individuals such as crackers. Threats from this source typically lack the resources of either structured or transnational external threats, but may nonetheless be very sophisticated. The motivation of such attackers is often, but not always, less hostile than that underlying the other two classes of external threat. Unstructured threat sources usually limit attacks to information system targets and employ computer attack techniques. New forms of loosely-organized virtual hacker organizations are emerging.

With a thorough understanding of the threats facing the system, organizations can better target their efforts in performing the risk analysis.

4.1.4.4 Architectural Risk Analysis

Once the documents and diagrams have been assembled, reviewers can begin the actual architectural risk analysis. Three activities guide architectural risk analysis—known vulnerability analysis, ambiguity analysis, and underlying platform vulnerability analysis [14]. By identifying and examining the conditions that must be met for vulnerabilities to be exploited and assessing the states that the system may enter upon exploitation, organizations can gain further insight into the security posture of the software entity.

- Vulnerability Analysis – There are a number of known vulnerabilities documented throughout software security literature. Known vulnerability analysis considers the architecture against a body of known bad practices or good principles for confidentiality, integrity, and availability.
- Ambiguity Analysis – Ambiguity is a source of vulnerabilities when it exists between requirements or specifications and development. Architecture's role is to eliminate potential misunderstandings between business requirements for software and developer implementation of the software's actions. Ambiguity analysis aims to identify where the requirements are ambiguously stated and the implementation and architecture disagree or fail to resolve the ambiguity.
- Underlying Platform Vulnerability Analysis – The goal of this step is to develop a list of application or system vulnerabilities that could be accidentally triggered or intentionally exploited and result in a security breach or violation of the system's security policy.

> When credible threats can be combined with the vulnerabilities uncovered in this exercise, a risk exists that requires further analysis and mitigation.

Hope, Lavenhar and Peterson further specify that the types of vulnerabilities that exist and the methodology needed to determine whether the vulnerabilities are present will vary depending on which phase in the SDLC the risk assessment occurs. In the requirements phase, the search for vulnerabilities should focus on the organization's security policies, planned security procedures, non-functional requirement definitions, use cases, and misuse and abuse cases. In the implementation phase, the identification of vulnerabilities should include more specific information, such as the planned security features described in the security design documentation. For fielded applications, the process of identifying vulnerabilities should include an analysis of the software security features and the security controls, technical and procedural, used to protect the system. Fielded systems can also use the results of system tests and reports from users to identify problems [14].

In addition, online vulnerability references should be consulted. Mailing lists, the National Vulnerability Database, and the vendor's website should provide information about existing vulnerabilities. These sites and lists should be consulted to make the vulnerability list current for a given architecture.

4.1.4.5 Organizing Risk Information

After completing the risk analysis, reviewers must organize the information such that informed risk mitigation decisions can be taken. Risk involves many components across assets, threats, vulnerabilities, and controls. Experience shows that the following questions help discussion participants understand the components of risk and uncover more information:

- What asset is being protected?
- How valuable is the asset to the organization?
- What threats (both known and potential) to the asset must be avoided?
- How might loss or exposures occur?
- What is the extent of potential exposure to the asset?
- What is the organization doing to reduce the probability or the extent of damage to the asset?
- What actions can be taken to reduce the probability of damage in the future?

A common pitfall in performing a risk assessment is to focus primarily on technology vulnerabilities. Experience shows that the most significant vulnerabilities often occur because of undefined processes or inadequate accountability for information security. Do not overlook the organizational and leadership aspects of security during the data-gathering process. For example, the inability to enforce updates on managed systems may lead to a breach of the integrity of information residing on those systems. Clear accountability and enforcement of information security policies is an issue in many organizations.

The combination of threats and vulnerabilities illustrates the risks that the system is exposed to. Shirey [15] provides a model of risks to a computer system related to disclosure, deception, disruption, and usurpation. Threats may target these risk classes:

- Disclosure – Dissemination of information to an individual(s) for whom the information should not be available
- Deception – Risks that involve unauthorized change and reception of malicious information stored on a computer system or data exchanged between computer systems
- Disruption –Access to a computer system is intentionally blocked as a result of an attack or other malicious action. It is important to note that in some cases, performance degradation can be as harmful as performance interruption
- Usurpation – Unauthorized access to system control functions.

Risk classification assists in the communication and documentation of risk management decisions. Threats and vulnerabilities conspire to participate in one or more risk categories; thus, mitigation mechanisms must also address one or more risk categories. Threats and vulnerabilities may combine to create additional weaknesses in the system.

4.1.4.6 Risk Likelihood Determination

Having determined what threats are important and what vulnerabilities might exist, it can be useful to estimate the likelihood of the possible risks. In software security, “likelihood” is a qualitative estimate of how likely a successful attack will be based on analysis and past experience. Because this estimate is based on past experience, this likelihood cannot account for new types of attacks or vulnerabilities that have not yet been discovered, and may not accurately reflect the probability of a successful attack. Nonetheless, the concept of likelihood can be useful when prioritizing risks and evaluating the effectiveness of potential mitigations.

The following factors must be considered in the likelihood estimation:

- The threat’s motivation and capability
- The vulnerability’s directness and impact
- The effectiveness of current controls.

The threat’s motivation and capability vary. Some vulnerabilities are direct and have severe impacts. For example, a vulnerability is direct and severe if it allows a database server to be compromised directly from the Internet using a widely distributed exploit kit. An indirect vulnerability that is less severe is one that requires an exploit payload to pass unmodified through several different systems only to produce a log entry that might cause an unexpected failure in the logging system.

The effectiveness of current security controls characterizes how high the bar is set for an intentional attacker or the likelihood of an accidental failure. For example, simple user IDs and passwords can be compromised more easily than most two-factor authentication systems. Adding a second authentication factor raises the bar for a would-be threat. However, if the second factor in the authentication is a biometric thumbprint reader that can be spoofed with latent image recovery techniques, the additional controls are not as effective.

The likelihood is a subjective combination of motivation, directness of vulnerability, and compensating controls. These factors culminate a rating of high, medium, or low. The likelihood levels are described in Table 10.

High	The three qualities are all weak. A threat is highly motivated and sufficiently capable, a vulnerability exists that is severe and direct, and controls to prevent the vulnerability from being exploited are ineffective.
Medium	One of the three qualities is compensating, but the others are not. The threat is perhaps not very motivated or is not sufficiently capable, the controls in place may be reasonably strong, or the vulnerability might be indirect or not very severe.
Low	Two or more of the three qualities are compensating. The threat might lack motivation or capability. Strong controls might be in place to prevent or significantly impede the vulnerability from being exploited. The vulnerability might be very indirect or very low impact.

Table 10: Risk Likelihood Levels

4.1.4.7 Risk Impact Determination

Independent of likelihood and controls, the risk impact must be determined. This impact is the consequences the business will face if the worst-case scenario in the risk description comes to pass. The analysis must also account for other credible scenarios that are not the worst case, yet are severe enough to warrant attention. The three aspects of risk impact determination are:

- Identify Threatened Assets – The assets threatened by the impact of this risk and the nature of what will happen to the assets must be identified. Common impacts to information assets include loss of data, corruption of data, unauthorized or unaudited modification of data, unavailability of data, corruption of audit trails, and insertion of invalid data.
- Identify Business Impact – The business will suffer some impact if an attack takes place. It is important to characterize the impact as specifically as possible. Risk management efforts are almost always funded by management in the organization whose primary concern is monetary. Management support and understanding can be assured only by driving software risks out to fiscal impacts.
- Impact Locality – All impacts will have a locality in space, time, policy, and law. In addition to characterizing the monetary impact, the location in other dimensions may be useful or required. For example, if an encryption key is stored unencrypted, it matters whether that key is in the dynamically allocated RAM of an application on a trusted server, on the hard disk of a server on the Internet, or in the memory of a client application. Likewise, laws and policies apply differently depending on where data is stored and how data exposures occur. Impacts may be localized in time or within business and technical boundaries.

Technical risk impact determination is supported by artifact analysis. There are a number of processes available for software risk identification, including the use of automated tools and the application of checklists and guidelines. The method used should strive to quantify risks in concrete terms. Examples of artifact quality metrics include, but are not limited to, the number of defects, the number of critical risks, identified risks by type, and progress against acceptance criteria.

As with risk likelihood, subjective high, medium, and low rankings may be used to determine relative levels of risk for the organization.

Two types of impact classes may have a more global impact. One is risks that may impact a domain system, such as a national or enterprise-wide system that is by its nature a single point of failure. The other concerns cascade failure, where failures in a technical system like the Domain Name Service may cascade across other systems and domains.

4.1.4.8 Risk Exposure Statement

The risk exposure statement combines the likelihood of the risk occurring with the impact of the risk. The product of these two sets of analyses provides the overall summary of risk exposure for the organization for each risk. Table 11 describes a method of generating the risk exposure statement.

		Impact		
		Low	**Medium**	**High**
Likelihood	**High**	Medium	High	High
	Medium	Low	Medium	High
	Low	Low	Low	Medium

Table 11: Risk Exposure Summary

The risk exposure statement generalizes the overall exposure of the organization for the given risk and offers more visibility to both impact and likelihood. The risk exposure statement gives the organization more control over risk management, but does not require all risks to be eliminated.

4.1.4.9 Risk Mitigation

The risks that have been identified and characterized through the process of risk analysis must be considered for mitigation. Mitigation of a risk changes the architecture of the software or the business in one or more ways to reduce the likelihood or impact of the risk. Formal and informal testing, such as penetration testing, may be used to test the effectiveness of the mitigations. Although changing how the business operates (e.g., insuring against impacts of risks) is a valid response to risk, it is outside the scope of architecture assessment, so it will not be covered here.

Mitigations to architectural flaws are almost always more complicated than mitigating implementation bugs. Addressing these flaws often requires cooperation between multiple modules, systems, or classes, and the cooperating entities may be managed and implemented by different teams. Thus, when a flaw is found, the fix usually requires agreement across multiple teams, testing of several integrated modules, and synchronization of release cycles that may not always be present in the different modules.

Reducing the likelihood of a risk can take several forms. Raising the bar in terms of the skills necessary to exploit a vulnerability is often a first step. For example, changing authentication mechanisms from user ID and password to pre-shared public key certificates can make it more difficult to impersonate a user. Reducing the period of time that a vulnerability is available to exploit is another way to reduce the likelihood of a risk. If sessions expire after 10 minutes of inactivity, the window of opportunity for session hijacking is about 10 minutes long.

Reducing the impact of a risk can take several forms. Most developers immediately consider eliminating the vulnerability altogether or fixing the flaw so that the architecture cannot be exploited. Cryptography can help when applied correctly. It is easier to detect corruption in encrypted data than in unencrypted data, and encrypted data is harder for an attacker to use if it is obtained. From a business point of view, it may make more sense to build functionality that logs and audits any successful exploits. Remediating a broken system may be prohibitively expensive, whereas adding enough functionality to have a high probability of stopping an exploit in progress might be sufficient.

Many mitigations can be described as either detection or correction strategies. Depending on the cost of making failure impossible through correction, it may be more cost effective to enable systems to detect and repair failure quickly and accurately. Imagine a software module that is temperamental and tends to crash when provided bad input and cannot be modified or replaced. A focus on correction would add business logic to validate input and ensure that the software module never received input that it could not handle. By contrast, a focus on correction would add monitoring or other software to watch for the module to crash and try to restart the module quickly with minimal impact.

Mitigation is never without cost. The fact that remediating a problem costs money makes the risk impact determination step even more critical. Mitigations can often be characterized in terms of their cost to the business—man-hours, cost of shipping new units with the improved software, delay entering the market with new features because old ones must be fixed, etc. However, the ability to characterize the cost of mitigation cost is of little value unless the cost of the business impact is known.

It is important to note that risk mitigation mechanisms may introduce threats and vulnerabilities to the system and will need to be analyzed. The risk analysis process is iterated to reflect the mitigation's risk profile.

4.1.4.10 When to Apply Risk Analysis

For an application that is in the initiation or design phase, information necessary to perform the architectural risk assessment can be derived from the design or requirements documents. For an application under development, it is necessary to define key security rules and attributes. System design documents and the system security plan can provide useful information about the security of software in the development phase. For software that has been fielded, data is collected about the software in the production environment, including data for system configuration, connectivity, and documented and undocumented procedures and practices. The system description is informed by the underlying security infrastructure or future security plans for the software.

In addition to reviewing the SDLC artifacts, questionnaires and interviews are useful in gathering information relevant to the risk assessment of the application. Policy documents, system documentation, and security-related documentation such as audit reports, risk assessment reports, system test results, system security plans, and security policies can also provide important information about the security controls used by and planned for the software.

In cases where the application is already in production or uses resources that are in production (e.g., databases, servers, and identity systems), these systems may have already been audited and assessed. These assessments, when they exist, may provide a rich set of analysis information.

Risk management is a continual process that regularly reevaluates business risks from software throughout the software's lifetime. Table 12 (taken from NIST SP800-34) describes the risk management activities that take place at various times during the lifecycle of a software system.

SLC Phase	Phase Characteristics	Risk Management Activities
Initiation	The need for software is expressed and the purpose and scope of the software is documented	Information assets are identified. Business impacts related to violation of the information assets are identified. Risks are considered in the system requirements, including non-functional and security requirements, and a security concept of operations is developed.
Development or Acquisition	The software is designed, purchased, programmed, developed, or otherwise constructed	The risks identified during this phase can be used to support the security analyses of the software and may lead to architecture or design tradeoffs during development.
Implementation	The system security features are configured, enabled, tested, and verified	The risk management process supports the assessment of the system implementation against the requirements and within the modeled operational environment. Decisions regarding identified risks must be made prior to system operation.
Operation or Maintenance	The system performs its functions. Typically, the system is being modified on an ongoing basis through the addition of hardware and software and by changes to organizational processes, policies, and procedures.	Risk management activities are performed for periodic system reauthorization (or reaccreditation) or when major changes are made to the software in its operational production environment (e.g., new features or functionality).

Table 12: Risk Management Activities defined by NIST

Due to cost, complexity, and other constraints, not all risks may be mitigated. Organizations may seek to accept the risk as a "cost of doing business." Risk management categorizes the controls that mitigate risks and tracks their efficacy over time through testing, log analysis, auditing, and other means. Risk measurement is a tool used to monitor the risk exposure of the organization over time. Metrics provide quantitative analysis information that may be used to judge the relative resilience of the system. Andrew Jaquith [16] provides guidelines that security metrics must adhere to:

- Be consistently measured – The criteria must be objective and repeatable
- Be cheap to gather – Using automated tools (such as scanning software or password crackers) helps

- Contain units of measure – Time, dollars, or some numerical scale should be included, not just traditional green, yellow, or red risks
- Be expressed as a number – Give the results as a percentage, ratio, or other actual measurement. Do not provide subjective opinions such as low risk or high priority.

Ongoing objective measurement provides insight into the effectiveness of risk management decisions and enables improvement over time. While the software industry as a whole currently lacks agreed-upon standards for precise interval scale metrics, software teams can adopt ordinal scale metrics that place events, controls, and security posture on a continuum. Ordinal scale metrics provide data that can be used to drive decision support by allowing visibility and modeling of the ranking of security metrics. Security metrics collection and analysis benefits from consistency. Although the measurements may emphasize specific aspects of the problem (counting lines of code to gauge complexity) while ignoring other aspects of the problem (interfaces to code), the trend data gained by using consistent measures remains valuable.

4.1.4.11 Tools

Unfortunately, there are no tools available that facilitate the understanding of a software architecture. However, graphically-based tools that support UML and reverse engineering of source code are useful in the architectural risk assessment process. These tools include:

- **Borland's Together –** This tool is a set of Eclipse plugins that provides UML 1.4 modeling, multi-language support, physical data modeling, design patterns, source code design pattern recognition, code template design and reuse, documentation generation, and code audits and metrics. Together adds language-neutral UML 2.0 diagramming, business process modeling, logical data modeling, logical-to-physical data model transformation, and custom pattern support. Borland’s strategy is to focus on tool integration, enabling best-of-breed strategies. Borland’s model-driven development (MDD) strategy is to offer application development teams a choice of tools and approaches rather than channeling them into a single modeling notation. Borland is active in the Object Management Group (OMG), leading the Graphical Model Framework (GMF) for Eclipse, and is also active in the Open Systems Group. Borland’s Together Visual modeling tool is based on UML2, BPMN, and MOF; it provides key features for software architects, designers, and coders. Together uses the OMG’s Query View Transformation (QVT) standard in model-to-model transformations and provides support for UML Object Constraint Language (OCL) 2.0. This includes audits and metrics, which are provided at both the model and code level and are defined in industry standard OCL. Overall, Borland provides a platform of common services and a set of practitioner tools within the broader scope of application life-cycle management.
- **AgileJ StructureViews –** This tool is a commercial Java visualization product that is deployed as an Eclipse Feature. The product brings together aspects of the Eclipse JDT Java model, set theory, class diagrams, and XP/Agile methods. The output resembles reverse-engineered CASE tool drawings. The Eclipse JDT model performs a number of functions in the Eclipse Java IDE, including populating the package explorer and type hierarchy trees. AgileJ StructureViews taps into that same source of information to populate its class diagrams. The visualizations are UML class diagrams, which can be printed or exported as JPEG images. Class diagrams appear alongside the source file

editor in the Eclipse IDE, and navigation is possible from any element on a diagram back to its source code. To comply with the XP goal of minimal documentation, no presentation-specific information is stored with a diagram. From the list of class names, all other information, including class members, inner classes, inheritances, associations and dependencies, is derived from Eclipse. The intention is that the diagrams only serve to increase comprehension of the coding model which they illustrate.

- **IBM's Rational Software Architect (RSA) –** This tool is a comprehensive modeling and development environment that leverages the UML for designing architecture for C++ and Java 2 Enterprise Edition (J2EE) applications and web services. RSA is built on the Eclipse open-source software framework and includes capabilities focused on architectural code analysis, C++, and MDD with the UML for creating resilient applications and web services. RSA provides visual construction tools to expedite software design and development and supports reverse transformations from Java to UML and C++ to UML. RSA enables model management for architectural re-factoring (e.g., split, combine, compare, and merge models and model fragments).
- **MicroGOLD's WithClass –** This tool is a feature-rich UML modeling tool. This product allows the user to draw UML diagrams, generate code, and reverse engineer popular object-oriented (OO) languages, including C++. Java, Delphi , VB, IDL, Perl, PHP, C#, and VB.Net. WithClass draws all UML type 1.x diagrams. VBA can be used to create add-ins to extend the functionality of the tool.
- **MagicDraw –** This tool is a visual UML modeling and CASE tool. This product facilitates analysis and design of object-oriented systems and databases. MagicDraw provides full support for J2EE, C#, C++, CORBA IDL programming languages, .NET, XML Schema, and WSDL, as well as database schema modeling, data definition layer (DDL) generation, and reverse engineering facilities. MagicDraw's reverse engineering capability allows generation of UML models from Java, C#, C++, CORBA IDL, EJB 2.0, DDL, CIL (MSIL), WSDL, and XML Schema source code. In addition, the product supports automatic generation of sequence diagrams from Java source code and adds a more detailed view of the system. MagicDraw's automatic report-generation engine produces comprehensive requirements, software design documentation, and other types of reports in HTML, PDF, and RTF formats. MagicDraw UML generates artifacts that match industry standard software development processes. The report engine allows users to generate up-to-date reports based on original templates with layout and formatting specified. MagicDraw runs on a wide variety of operating systems, including Windows 98/ME/NT/2000/XP/Vista, Solaris, OS/2, Linux, HP-UX, AIX, MacOS (X), and other platforms that support Java 5 and 6.
- **Sparx System's Enterprise Architect –** This tool is a comprehensive UML analysis and design tool. Enterprise Architect provides complete traceability from requirements analysis and design artifacts through to implementation and deployment. Enterprise Architect is built upon the UML 2 specification. This product can display UML profiles to extend the modeling domain and includes model validation to ensure integrity. This tool combines business processes, information, and work flows into one model using extensions for BPMN and the Eriksson-Penker profile. Supported diagrams include object, composite, package, component, deployment, use case, communication, sequence, interaction, activity, state, and timing.

- **Visustin Flow Chart Generator –** This tool is an automated program for generating flow charts. Visustin can be used to create flowcharts and UML activity diagram-style charts from source code. Visustin accepts source code files as input and supports a total of 31 programming languages. Supported languages include Ada, ASP, assembler, BASIC, C/C++, C#, Clipper, COBOL, ColdFusion, Fortran, Java, JSP, JavaScript, LotusScript, Pascal/Delphi, Perl, PHP, PL/SQL, PowerScript, PureBasic, Python, QuickBASIC, REALbasic, T-SQL, VB, VBA, VB.NET, Visual FoxPro, and XSLT. Visustin creates a flow chart or an activity diagram using automated layout routines. The user can also draw a flow chart manually. The resulting flow chart can be printed, saved in multiple bitmap and vector formats, or exported to external programs (e.g., Microsoft Word, Visio, and PowerPoint). Visustin provides several source code metrics, including McCabe cyclomatic complexity and Decision density.

4.1.5 Pedigree Analysis

Many organizations are taking advantage of open source software, which can provide cost savings and other improvements by relying on a (potentially) well-tested and robust code base. In addition to leveraging open source software, organizations may extend the software to customize it for their own internal purposes. This provides a number of benefits to the organization, including reducing development time.

The trend of embracing open source software has led to concern within the software community. One of most common concerns facing organizations is that developers or projects may incorporate open source software without the full backing of an organization’s legal department, possibly resulting in future licensing issues. Similarly, many organizations are may be unaware of the pedigree of the source code within their own software-intensive systems. From a security—and potentially safety—perspective, this may result in organizations foregoing important patches for open source software that has been co-opted into a larger organizational software base.

A number of vendors have developed pedigree analysis tools, which can scan software within an organization and highlight its original source. Any software from an external source (most commonly open source software) can be correlated to a project name and version number against which licensing and patching concerns can be resolved. Beyond identifying external source code, pedigree analysis tools can be useful in identifying the components of the code that have been custom developed. These components may require more thorough testing than the external source code.

4.1.5.1 When to Use Pedigree Analysis

Pedigree analysis tools do not directly scan for vulnerabilities in the way that static analysis scanners do. Instead, pedigree analysis tools aim to improve an organization’s security posture by identifying non-developmental components and providing insight into any unmitigated security vulnerabilities that may be introduced by these components.

Because they work directly against the source code, pedigree analysis tools can be used at any point in the development lifecycle where source code artifacts are available. As with any analysis of source code, it is best to begin performing the analysis as soon as code is being generated. Unlike with static analysis tools, pedigree analysis tools do not need to be integrated with development tools or even used by developers. Instead, they can be incorporated into regular audits of the source to identify new or questionable material within the source. Even in situations where an organization cannot deploy the tool until later in the software's lifecycle, the results from a pedigree analysis tool can still be important in understanding an organization's security posture.

4.1.5.2 Benefits of Pedigree Analysis

Static analysis tools have the following benefits:

- Reducing costs associated with analyzing the source – Separating non-developmental code from custom developed code reduces the costs associated with reviewing the software for vulnerabilities
- Reducing costs over the system lifetime – Vulnerabilities from incorporated components identified early in the lifecycle are cheaper to fix.
- Educating developers – In some cases, developers may be unaware of the consequences of incorporating and extending external code into a system (e.g., some organizations may be maintaining their own forked version of an existing software code base, increasing the difficulties associated with patch management).

4.1.5.3 Drawbacks of Pedigree Analysis

Pedigree analysis tools have limited use in situations where a large portion of the code base is custom developed or relies on less-popular open source software. Not all open source projects receive equal scrutiny in regards to security analysis, potentially resulting in vulnerabilities that may not be disclosed by a pedigree analysis tool. Similarly, open source projects focusing on small communities of contributors and users may not be included in pedigree analysis tool database.

4.1.5.4 Pedigree Analysis Tools

There are currently few tools that perform pedigree analysis. Some of the primary tools include Palamida, Blackduck Software, and the open source EULAyzer tool.

- **Palamida –** This tool automates scanning source code and finds intellectual property and licensing issues. Palamida offers additional services aimed at resolving security defects occurring in externally developed software. *http://www.palamida.com/*
- **Black Duck Software –** Automatic tool for scanning source code and finding intellectual property and licensing issues. The analyses are performed based on patterns of licenses and known packages (e.g., SF.net). *http://www.blackducksoftware.com/index.html.*
- **EULAyzer –** Open source tool for analyzing license agreements to gain a better understanding of how the software can be used. *http://www.javacoolsoftware.com/eulalyzer.html.*

4.2 Analysis of Executables

This section will provide an overview of information on the techniques available for analyzing executables. These techniques will be described in detail in the following subsections.

4.2.1 Binary Code Analysis

Binary code analysis automates the process of reviewing binary code for security- or safety-related issues. One of the goals for binary analysis tools, as identified in *Safety Checking of Machine Code* [17], is to allow for more freedom in developing the application rather than forcing safe (or secure) libraries or source languages on the developer. This is increasingly important in an environment where the source code is unavailable because binary code may be the only construct to be examined.

Binary code analysis tools are similar to disassembler analysis tools. Both types of tools perform analysis on the most readily available software artifact—the binary executable. The primary distinction between these two forms of tools is that binary analysis tools directly analyze the binary opcodes (or their assembler equivalents) of the software rather than the higher-level instructions they may represent. Due to the effort required to perform such analysis, there are few commercially available tools that perform this type of analysis. The @stake SmartRisk Analyzer, introduced in 2004, directly analyzed binary executables for potential vulnerabilities. In part due to the expensive computational requirements it imposed, commercial availability of the SmartRisk Analyzer was short-lived. The underlying technology of the SmartRisk Analyzer powers the security analysis service offered by Veracode.

4.2.1.1 When to Use Binary Analysis

Binary analysis tools are commonly used at the end of the software development lifecycle. Often, the goal is to determine the security or safety of COTS products prior to purchase. This is important because many COTS vendors are unwilling (or unable, due to licensing agreements) to offer the source code for review, leaving binary analysis as the only way to scan the application.

For binary analysis tools that are offered as a service (e.g., those offered by Veracode), it may be impractical for an organization to deploy binary analysis against its own internally-developed software—particularly with the availability of source code analysis tools. Nevertheless, the binary represents the final and definitive version of the software. While static analysis can prove useful, the source code does not represent the exact sequence of actions that will be performed by the software. In contrast, binary analysis has the potential discern security vulnerabilities that may not exist within the code itself.

4.2.1.2 Required Skills

With the relative dearth of pure binary analysis tools—Veracode is the primary vendor in this space—there are few skills required when using this technique. For example, Veracode users need only upload the target binary files to the Veracode Website. In contrast, some of the more advanced tools being researched (e.g., those discussed in academic papers) require a more indepth understanding of binary opcodes and their assembler equivalents to truly understand the results of the tool or even what is occurring within the application itself.

4.2.1.3 Benefits

Binary analysis tools have the following potential benefits:

- No need for source code – Binary analysis tools would allow COTS applications to be fully analyzed without access to vendor-provided source code or documentation
- No disassembly – The primary difference between pure binary analysis tools and disassembler-based tools is the need to disassemble the binary itself. In many situations, this is prohibited by the licensing agreement in place with the vendor. As such, pure binary analysis allows for thorough exploration of the application without violating licensing agreements.

4.2.1.4 Drawbacks

As mentioned above, one of the primary drawbacks associated with binary analysis tools is the relative lack of availability. Predicting the execution of a large binary application is a hard problem and is currently the focus of a number of research projects. In addition, existing binary analysis tools require that the application be compiled with debugging enabled, providing very important information about variables and controls paths within the binary itself. This requirement can be problematic because most applications are compiled without debugging enabled for size and performance optimization.

4.2.1.5 Specific Tools and Services

There are a limited number of binary analysis tools available on the market as the academic community is focused mostly on prototypes while there is one primary COTS vendor in this field. These two sources are listed below:

- **Thesis-ware** – There are a number of research projects that take advantage of binary analysis. Unfortunately, the majority of this research has yet to leave academic settings and has not progressed beyond the initial academic papers or thesis in which the tools are introduced. One such example is in *Safety Checking of Machine Code* [17]
- **Veracode** (formerly @stake's analyzer) – Incubated by Symantec and launched as its own company in 2006, Veracode offers third-party analysis of binary executables. By offering their binary analysis as a service, Veracode's tools can run with the large amount of resources consistent with the difficulty of binary analysis. For comparison, @stake's SmartRisk Analyzer required one gigabyte of memory in 2004. In addition, Veracode's offering of binary analysis as a service allows them to cooperate with COTS vendors in ways that would be unavailable to direct customers, much like traditional testing laboratories. *http://www.veracode.com/*.

4.2.2 Disassembler Analysis Tools

Disassembler analysis tools are similar to binary analysis tools in that they automate the process of reviewing the binary for security or safety concerns. However, disassemble analysis tools provide intermediate representations (or high-level language representations) of the binary to be analyzed. As such, disassembler tools can provide a higher level of semantic information than the binary itself.

The majority of current tools and techniques are used for:

- Standards compliance analysis to ensure coding of the source followed coding standards
- Reengineering analysis prior to migration of legacy code from an old host (e.g., mainframe) to a new host.
- Safety analysis to ensure that specific conditions have been met.

The methodology in place for using existing tools for safety and security assessment of disassembled code first relies on performing a successful disassembly and, in some instances, decompilation to a high-level language. The primary tool used in the research and development (R&D) community is Hex-Rays' Interactive Disassembler (IDA). IDA is a static, interactive disassembler that targets the majority of processor and assembler languages. Due to the high-interaction required by IDA, many organizations have developed tools to automate this process. One such example is Grammatech's CodeSurfer/x86, which is an extension to IDA.

4.2.2.1 When to Use Disassembler Analysis

Like binary analysis tools, disassembler analysis tools are commonly used at the end of the software development lifecycle. There is a good deal of ambiguity introduced by disassembling binary code. In an environment where source code is available, directly analyzing the source code may be preferred over disassembler analysis. The binary code is the final output of the application and some organizations may need to perform analysis of the results of compilation in addition to the source code itself. Nevertheless, these tools lend themselves to being deployed at any phase in the lifecycle in which binary code is available.

4.2.2.2 Benefits

Disassembler analysis tools have the following benefits:

- No need for source code – Ideally, binary analysis tools would allow COTS applications to be fully analyzed without access to vendor-provided source code or documentation
- Guaranteed analysis – Because the analysis is performed on the final output, organizations can be guaranteed that the binary code scanned is the same as the binary code that will be run.

4.2.2.3 Drawbacks

As mentioned above, one of the primary drawbacks associated with disassembler analysis tools is their need to disassemble existing binaries, which can run counter to the license agreement through which the binaries may be made available. These licensing issues can sometimes be addressed. In some cases (e.g., analyzing malicious code or custom-developed software) licensing issues may not be a concern. Nevertheless, the reliance that many of these tools have on a single major tool—IDA—leaves them vulnerable to issues facing the base tool. In particular, IDA has known issues where it may not accurately analyze the underlying binary, potentially resulting in the propagation of incorrect information through the rest of the analysis. Similarly, IDA's reliance on manual review of the underlying binary can lead to human error propagating throughout much of the analysis of the software system.

4.2.2.4 Specific Tools and Services for Hire

A large number of tools are available for disassemble analysis. The following list highlights the majority of analysis tools available for use:

- **IDA Disassembler and Debugger –** Offered by Hex Rays, IDA is one of the most popular disassemblers in use. IDA provides a large number of features for analyzing and disassembling binary code. In addition, Hex Rays offers older versions of IDA Pro free for non-commercial use. *http://www.hex-rays.com/idapro/*
- **Codesurfer –** An extension to IDA pro that aids in the analysis of C and C++ binaries. Built on the Wisconsin Program-Slicing tool, CodeSurfer provides call graphs and other analysis tools for improved understanding of source code. *http://www.grammatech.com/products/codesurfer/overview.html*
- **Boomerang –** An open source project that aims to offer an alternative to IDA. Boomerang development was halted in 2006, but its source code remains available. *http://boomerang.sourceforge.net/*
- **Wisconsin Safety Analyzer –** University of Wisconsin's WiSA project offers a number of tools that extend IDA Pro and Codesurfer, specifically for safety analysis. Specific projects include the Sandia BREW project [18]. *http://www.cs.wisc.edu/wisa/*
- **OpenRCE.org –** A website offering community-developed plugins for the IDA Pro disassembler. *http://www.openrce.org/*
- **Virtutech –** The Simics Hindsight reverse execution debugger aids organizations in the analysis of binary executables, specifically for embedded systems. *http://www.virtutech.com/products/simics_hindsight.html*

4.2.3 Binary Fault Injection

Like source code fault injection, binary fault injection was originally developed by the software safety community to reveal flaws that occur only when certain failure conditions are present. While source code fault injection instruments the code such that faults will be forced to occur, binary fault injection focuses on likely faults that will occur in the real-time operation of the software (e.g., memory faults or other error conditions provided by the processor). In the safety community, the goal is to ensure that these faults do not result in unsafe execution of the software. In the security community, the goal of these faults is to simulate anomalies, attack patterns, or unintentional faults that would lead the software to enter a vulnerable state.

Binary fault injection is similar to fuzz testing. The primary goal is to induce the software to enter a state through the application of invalid data. Binary fault injection accomplish this through the application of error conditions that software may receive rather than specifically providing invalid data to the software. Environment faults are particularly useful to test because they are likely to reflect real-world attack scenarios. Because of the complexity of the fault injection testing process, it tends to be used only for software that requires very high confidence or assurance.

As mentioned in Section 4.1.2, the goal is to analyze the effects of a single fault within the software through the application as a whole. Using this information, the tester can extrapolate the impact of a particular fault on the software as a whole that can lead to exploitable vulnerabilities.

4.2.3.1 When to Use Binary Fault Injection

Binary fault injection tools can be used at any point in the development lifecycle where binary artifacts are available. As with source code fault injection, the full benefits of binary fault injection may not be fully realized until a substantial portion of the software application is available for integration or unit testing because one of the primary aspects of all fault injection is in fault propagation analysis. Organizations may benefit from performing binary fault injection in a simulated deployment environment to better test the effects of likely faults on the system. As with any software analysis tool, it is important to deploy binary fault injection as early in the software development lifecycle as possible.

4.2.3.2 Required Skills

Unlike source code fault injection, many fault injection tools are targeted at users without a deep understanding of the fault injection process or a strong understanding of the code base against which testing will be performed. Many of the commercial vendors provide simple user interfaces and test scripts that can be run against the target system. As with any testing technique, the quality of analysis and mitigation strategies will be affected by the reviewer's background. Nevertheless, important findings can be identified by injecting simple binary faults into the system (e.g., memory or input/output faults, which are common in a deployment environment).

4.2.3.3 Benefits

Fault injection tools have the following benefits:

- Increased test coverage – While not to the level of coverage offered by source code fault injection, binary fault injection tools can simulate events and environmental conditions that traditional testing methods may be unable to replicate
- Repeatability – By simulating the effects of environmental faults on the application, reviewers can ensure that results are repeatable. If these conditions occur in a deployed environment, it may be difficult to repeat the effects to determine the exact cause of the issue.

4.2.3.4 Drawbacks

Binary fault injection tools require users to perform more detailed fault propagation analysis to understand the root cause of errors resulting from supplied faults. Unlike source code fault injection, reviewers are unaware of the exact point at which the fault was injected and may need to have a very thorough understanding of the underlying code base. Some fault injection tools resolve this issue by combining fault injection with dynamic analysis, allowing reviewers to follow the flow of the software throughout the testing cycle.

4.2.3.5 Tools

A number of these tools also support source code fault injection, but their primary stated purpose is for binary fault injection. It is interested to note that few, if any, open source binary fault injection tools are available on the market. The list below highlights the three primary binary fault injection tools identified in this review:

- **Holodeck –** A binary fault injection tool with security-specific capabilities. Through a mixture of fault injection and fuzz testing, Holodeck aims to provide improved security testing through its dynamic analysis capabilities. *http://www.securityinnovation.com/holodeck/*
- **Exhaustif –** A binary fault injection tool that simulates many of the errors that may occur while software is deployed. Specifically aimed at embedded systems and control systems, which must have high fault tolerance, Exhaustif has been extended to support distributed systems. *http://www.exhaustif.es/*
- **Xception –** A binary fault injection tool that simulates an embedded processing environment. Xception provides support for a variety of operating systems, including Windows 2000, LynxOS, and Linux. *http://www.xception.org/*

4.2.4 Fuzzing

Fuzzing is described in “Enhancing the Development Lifecycle to Produce Secure Software” [19] as supplying “random invalid data (usually produced by modifying valid input) to the software via its environment or another software component.” Fuzzing is often implemented via a “fuzzer,” a program or script that submits a combination of inputs to the software to reveal how that software responds. Fuzzers are generally specific to a particular type of input, such as HTTP input, and are written to be used to test a specific program; as such, they are not easily reusable.

Takanen et al. [20] describe fuzz testing in detail in their book, *Fuzzing for Software Security Testing and Quality Assurance*. Fuzz testing is a form of negative testing. In negative testing, reviewers verify that the system meets requirements by providing it with invalid data (e.g., supplying an eight character password to a system requiring ten characters). Fuzz testing takes this a step further by introducing randomness.

4.2.4.1 When to Use Fuzzing

Fuzzing tools can be used at any point in the development lifecycle. In fact, various fuzzing tools fit better in different points in the development lifecycle. Some fuzzing tools can be used in tandem with unit testing, which is performed whenever developers complete a portion of the software and prior to committing changes to the source code repository. Other fuzzing tools are used during integration testing—analyzing the effects of fuzzed input on the various components that comprise the software system. Further fuzzing tools can be used to analyze COTS products, either before selection or after deployment, to determine the security properties and robustness of these tools before they are integrated with the rest of the software-intensive system. Fuzzing tools can also be used on deployed software to identify existing issues so that mitigation strategies can be crafted.

4.2.4.2 Required Skills

As outlined in [19], effective fuzz testing requires the tester to have a thorough understanding of the software being tested and how it interfaces with external entities whose data will be simulated by the fuzzer. Testers would benefit from a thorough understanding of the concepts behind fuzz testing. A number of resources are available (including [20]) on the Internet.

Most fuzz testing tools are provided as a framework on which a specific tool can be built. Many of the techniques and methods for generating fuzzed data are provided by these frameworks. Testers are expected to provide the “glue code” that will interface directly with the software application. Some fuzzer frameworks are generic, which is useful for developing highly complex fuzzing test cases, but requires detailed knowledge of the interfaces being tested and the framework itself. More dynamic fuzzers are also available. These fuzzers “learn” the interface between the software and the test tool by observing valid communication. Perturbations are then added to the valid communication to introduce fuzzed data into the system.

When performing fuzz testing, testers need to have the appropriate expertise to diagnose errors within the application. The results of random inputs may not immediately manifest themselves within the system. Some data may be inadvertently stored as state information within the application’s memory and will affect its behavior in an unknown fashion.

4.2.4.3 Benefits of Fuzzing

Fuzz testing provides the following benefits:

- Increased code coverage – By introducing random information into testing, organizations can ensure wider test coverage beyond that of pure functional testing
- Corner cases – Random inputs can provide insight into how corner cases affect the system. These can aid in identifying portions of the application that may have logic errors
- Additional test cases – Through the use of random inputs, test cases that would not be covered by methodical, deterministic testing can be introduced
- Repeatable – While fuzz testing introduces a random element to security testing, it is nevertheless repeatable. Recording the input sent to the system or storing the random seeds used to generate the tests will ensure that testing is repeatable.

4.2.4.4 Drawbacks

The primary drawback of fuzz testing lies in the complexity of analyzing its results. In many cases, it may be necessary to use dynamic analysis or a debugger to follow the execution of the random input. Random input can result in program flow that is contrary to design or expectations. Similarly, reviewers may face difficulty in correctly crafting a fuzzer for the application itself. Providing random input, while possibly important, is usually insufficient for fuzz testing. In most applications, a certain level of invalid data will cause the entire stream of input to be rejected, resulting in a waste of effort on the part of the reviewers.

4.2.4.5 Tools

As mentioned in Section 4.2.4.2, the majority of fuzzing tools are available as a framework that must be extended and customized for each application being tested. As such, the majority of fuzzing tools publicly available are open source, allowing organizations to easily extend and adapt the framework for their own needs. These tools include:

- **SPIKE –** A fuzzing framework that allows users to develop fuzzing tools tailored to their applications or environments. *http://www.immunitysec.com/resources-freesoftware.shtml*
- **Sulley –** A fuzzing framework that allows users to take advantage of multiple extensible components. Sulley provides tools for data representation, data transmission and target monitoring. *http://www.fuzzing.org/wp-content/SulleyManual.pdf*
- **GPF –** An evolutionary fuzzer that attempts to “learn” protocols prior to generating fuzz test cases against the target software. *http://www.vdalabs.com/tools/efs_gpf.html*
- **Peach Fuzzing Platform –** Peach provides tools for generation and mutation-based fuzzing with a large community of developers and users. *http://peachfuzzer.com/*

4.2.5 Malicious Code Detectors

Malicious code detectors find the hallmarks of malicious logic embedded in programs. This is contrast to software aimed primarily at finding delivered malware (e.g., viruses, worms, spyware, etc.). In some cases, the techniques and methods used to perform malicious code detection may be similar to reverse engineering or dynamic analysis techniques. However, instead of identifying the security properties of the binary being analyzed, malicious code detectors aim to make a determination as to whether the binary is inherently malicious.

These tools are still in the early phases of research and development and are not yet robust enough to be incorporated into an organization’s development environment. Some organizations have begun taking advantage of virtualization and virtual machine introspection (where the operation of the virtual machine is monitored from outside of the machine itself) to determine whether software is malicious. This is in contrast to the detectors discussed in this section, which focus more specifically on monitoring the binary itself.

4.2.5.1 Required Skills

In this field’s current state, the majority of tools require some understanding of the concept. Many of these tools are only available in the academic community and are still under development. However, the goal of these tools, however, is to automate the process of malicious code detection, which would result in fewer skills required on the part of the security analyst.

4.2.5.2 Benefits

The primary benefit associated with malicious code detectors is the ability to determine, through an automated process, the intent of a supplied binary application. Many concerns associated with malicious code detection revolve around the fact that malicious code purposefully or inadvertently inserted into a software application does not, at first glance, appear to be malicious. These tools aid reviewers in identifying and understanding obfuscated logic that would identify the existence of malicious code within an application.

4.2.5.3 Drawbacks

Malicious code detectors are currently academic oriented and much of the necessary research is still under way. As such, tools such as CERT's functional extraction tool are only effective against a small subset of malicious code that may exist in a software-intensive system. Nevertheless, malicious code detectors have the potential to be important aspects of any software security testing regimen.

4.2.5.4 Tools

The two primary malicious code detectors identified in this review are:

- **Fakebust Fake Exploit Code Detector –** Fakebust assists with the rapid assessment and supervised execution of potentially malicious programs such as exploits or utilities of unknown origin or programs recovered during operating system forensics. Fakebust provides a sandboxed environment in which the potentially-malicious code can be run. When the application running within Fakebust begins to request permissions or perform actions that are outside Fakebust's accepted boundaries, it will inform the user. *http://www.securiteam.com/tools/6R0061FBFY.html*
- **Functional Extraction –** CERT's functional extraction tool aims to take advantage of the power of reverse engineering. Unlike some of the reverse engineering tools currently in use, functional extraction uses function-theoretic concepts to automate understanding of the behavior of the program being examined. One of the goals of the project is to more readily bypass some of the obfuscation techniques that occur in malicious code. *http://www.cert.org/sse/function_extraction.html*

4.3 Analysis of Intermediate Representations

The objectives of bytecode, assembler, and object code analyses will be discussed here. This section will also include the merits of each compared with the others (including source code and binary code analysis) and analysis of the running system (e.g., penetration testing).

4.3.1 Bytecode Analysis

Bytecode is an intermediary form of a software executable that is designed to be processed by an interpreter or compiler before being run directly on machine hardware. The most common examples of bytecode occur in Java applications and Microsoft .NET's Common Intermediate Language (CIL). As an intermediary form of executables, bytecode contains more semantic information about the program's execution than an equivalent binary. As such, bytecode has proven to be a powerful medium against which analysis can be performed.

The main purpose of after development bytecode analysis tools (vs. tools built into runtime code interpreters) is to establish that the software exhibits properties consistent with quality objectives. There are no bytecode analysis tools specifically geared towards verifying and validating software security or safety properties. There is a significant amount of work being performed in the R&D communities to develop bytecode analysis tools aimed at security and safety analysis, including defining repeatable methodologies for using current tools. Some security and safety analysts are already using these tools.

The following subsections include tools that have been successfully used for these types of analyses when limited to specific foci (e.g., verifying correctness of security functions). The majority of these tools focus specifically on Java bytecode because it was introduced with the Java language in the 1990s. Bytecode for many other commonly used development languages has been recently introduced (e.g., CIL was introduced in 2002 with Microsoft's Visual Studio .NET package)

Bytecode scanners provide additional capabilities over pure binary or pure source code-oriented analysis tools. Bytecode is similar to binary code. It provides a direct representation after optimization of specific actions to be performed by the application. In addition, bytecode allows for more information about the intent of the application than pure binary opcodes can provide. As shown in "Java Bytecode," [21], bytecode can represent information about coding style and details about the efficiency of the original high-level code.

4.3.1.1 When to Use Bytecode Analysis

Bytecode analysis tools are commonly used at the end of the software development lifecycle. However, in many cases, the source code is also available, allowing bytecode analysis and source code analysis to occur in tandem, improving the accuracy of the results. It is important to note that these tools can be deployed at any phase in the lifecycle in which bytecode can be generated.

4.3.1.2 Required Skills

The expertise required for bytecode scanners varies. The majority of these tools do not require users to be fully versed in a language's bytecode, but a basic understanding of the language's bytecode may be necessary to fully analyze the tool's output.

4.3.1.3 Benefits

Bytecode scanners provide the following benefits:

- Improvements over source code analysis – By integrating source code analysis with bytecode analysis, a number of tool vendors can improve false positive and false negative rates associated with a particular language
- Improved accuracy – By analyzing the bytecode rather than the high-level language, tools can focus on a tighter, more restricted program flow
- Checking for good programming style – Because a good deal of program style can be identified through the bytecode, these tools can offer support for programming style verification.

4.3.1.4 Drawbacks

Unlike other tools, bytecode analysis tools are strictly limited to software implemented in the target language, and the majority of these tools only focus on a single target bytecode—Java or .NET. With the increasing performance benefits that bytecode-based languages are seeing, other languages have begun supporting their own forms of bytecode. New tools will need to be developed to allow organizations to leverage bytecode analysis in environments deploying these new languages. In the future, it is likely that many bytecode analysis scanners will evolve to support a larger subset of languages than their current form.

4.3.1.5 Open Source Tools

The following open source tools and frameworks are available for bytecode analysis:

- **ObjectWeb ASM** – This tool performs Java bytecode manipulation and framework analysis. This tool can be used to modify existing Java classes or to dynamically generate classes directly in binary form. The framework includes common transformations and analysis algorithms to enable easy assembly of custom-defined complex transformations and to custom-generate code analysis tools. This tool is being used by several research projects interested in safety or security analysis of Java bytecode. *http://asm.objectweb.org/*
- **Package javassist.bytecode.analysis (Javassist API)** – The Java Programming Assistant (Javassist) tool includes an API for performing data-flow analysis on a Java method's bytecode, enabling the analyst to determine the state of the stack and local variable table at the beginning of each instruction. This API can also be used to validate bytecode, find dead bytecode, and identify unnecessary checkcasts. *http://www.csg.is.titech.ac.jp/~chiba/javassist/html/javassist/bytecode/analysis/package-summary.html* and *http://www.csg.is.titech.ac.jp/~chiba/javassist/*
- **FindBugs** – This tool is an open-source static bytecode analyzer for Java class files (based on Jakarta BCEL). FindBugs is used to check for null pointer deferences, synchronization errors, vulnerabilities to malicious code, etc. The tool can be used to create linkages between the bytecode and the original Java source code to highlight the location of the causal problem in the source code. *http://findbugs.sourceforge.net/* and *http://en.wikipedia.org/wiki/FindBugs*
- **BCEL** – The Byte Code Engineering Library is an API developers can use to analyze, create, and manipulate Java byte code. It is the basis for automated bytecode analyzers, including FindBugs. *http://jakarta.apache.org/bcel/.*

4.3.1.6 Commercial Tools

The following commercial tools and frameworks are available for bytecode analysis:

- **StackFrame, LLCTorqueWrench –** This static Java bytecode analysis tool looks for violations of coding standards and practices and some coding problems. This tool is probably not relevant for the applications being reviewed for this paper. *http://www.stackframe.com/TorqueWrench/*
- **IBM Security Research Java Bytecode Analysis (JABA) –** JABA is a component of IBM's AlphaWorks Security Workbench Development Environment for Java (a.k.a. SWORD4J). SWORD4J is a set of Eclipse plug-ins that collectively perform static analysis of Java programs. *http://www.research.ibm.com/javasec/JaBA.html* and *http://www.alphaworks.ibm.com/tech/sword4j*
- **Sun Microsystems JFluid –** This tool requires the instrumentation of the bytecode or object code that will be analyzed by the tool. This tool is still considered an R&D application and is not yet ready for extensive use. *http://research.sun.com/techrep/2003/smli_tr-2003-125.pdf.*

According to NIST's SAMATE site, the following COTS tools also perform bytecode analysis. Based on the tool and vendor webpages, it is difficult to determine if the tools perform the necessary analyses, and if so, whether the analyses are relevant for security and safety. Tool descriptions are based on the info provided by SAMATE.

- **SofCheck Inspector –** This tool creates assertions for each module to prove system assertions and the absence of runtime errors. This tool works with Java. *http://www.sofcheck.com/products/inspector.html*
- **Microsoft - FxCop –** This tool checks .NET-managed code assembly conformance to Microsoft .NET framework design guidelines. Microsoft claims the tool looks for more than 200 defects in library design, globalization, naming conventions, performance, interoperability, portability, security, and usage. *http://code.msdn.microsoft.com/codeanalysis/Release/ProjectReleases.aspx?ReleaseId=553.*

4.3.1.7 Proprietary Tools Used by Commercial Services Providers

The following tools do not appear to be available for purchase or licensing. They are used by service providers to do analyses for hire.

- **OPTIMA Business Information Technology GmbH OPTIMA Bytecode Scanner –** This tool is used in OPTIMA's code review services. The tool performs pattern and data flow analyses, and is designed to detect the maximum number of security problems (even at the risk of generating more false positives). For example, the tool can be modified to perform a validation that "untaints" an object. OPTIMA has used this tool to analyze cryptography and other security-specific functional implementations. The tool is scalable for large projects (large code bases) and has been used by OPTIMA on a number of such code reviews. The tool is not fully automated. OPTIMA relies on expert human analysis assisted by the tool. *http://www.optimabit.com/en/optima/home.html*

- **Aspect Security AspectCheck –** Aspect Security uses AspectCheck in code review services to check for security-critical calls in Java and .NET applications, including ASP.NET, C#, and VB.NET. *http://www.aspectsecurity.com/documents/Aspect_S3_Securing_J2EE_Services.ppt*
- **Trusted Labs TL ADT –** This tool is used as part of Trusted Labs' security evaluation services (including white box (source code) static analysis). The Trusted Labs analyst runs TL ADT to automate code reviews in connection with certification and accreditation (C&A) using TL ADT. The tool is used to challenge the Java application bytecode against a set of security rules defined by or for the certification authority. Implementing the rules inside TL ADT allows the Trusted Labs analyst to examine the source code. *http://www.trusted-labs.com/se_services.html.*

4.4 Supporting Technologies

While tools are an important aspect of any software security testing regimen, there are a number of technologies currently available and under development that can aid organizations and reviewers. Organizations can take advantage of test case generators, which will aid in preparing security test plans for software; test oracles, which provide information about the expected results for specific tests; and other technologies. This section describes a subset of these technologies and how they can be used within a security testing framework.

4.4.1 Tool Integration and Normalization

In 2006, Kris Britton of the National Security Agency (NSA) Center for Assured Software (CAS) observed that the level of integration of software security tools has not reached a point where these tools can support "meta-analysis," where tools can incorporate the results of one another and interpret, rank and adjust confidence levels appropriately [22]. Most security testing tools cover only a small subset of test patterns or vulnerabilities associated with software.

To aid the ability of tools to work together, the OMG Software Assurance Special Interest Group is developing the Software Assurance Ecosystem. The ecosystem is a suite of OMG specifications that can be leveraged together to improve interoperability among tools. These specifications include:

- Knowledge Discovery Metamodel (KDM) – This specification provides an ontology that can be used to describe key aspects of software, improving interoperability among tool outputs.
- Semantics of Business Vocabulary and Rules– This specification provides the basis for a formal, natural language interpretation of compliance rules, security policies, or other criteria against which the software will be analyzed.

Hatha Systems offers a set of tools, KDM Analytics, that implement these OMG specifications. While these tools are not yet offered as standalone COTS products for testers, Hatha Systems aims to foster the development of these specifications with the goals of increasing interoperability among major tool vendors.

4.4.2 Vulnerability Categorizations and Markup Languages

Over the past few years, a number of vulnerability classification and markup language efforts have emerged. To reign in the effects of competing standards and specifications, NIST launched the Information Security Automation Program (ISAP) and the SCAP. Through ISAP and SCAP, NIST has identified a set of vulnerability classifications and markup languages that form the basis for SCAP-support. Currently, the National Vulnerability Database offers services that comply with many of these standards.

The major SCAP standards are:

- Common Vulnerabilities and Exposures (CVE) – CVE provides a standard name and identifier for individual vulnerabilities and exposures that have been publicly identified
- Common Configuration Enumeration (CCE) – CCE provides a standard name and identifier for individual configuration issues associated with software components
- Common Platform Enumeration (CPE) – CPE provides a standard name and identifier for specific systems, platforms, and packages
- Common Vulnerability Scoring System (CVSS) – CVSS provides a metric for quantitatively communicating the impact of a specific vulnerability
- Extensible Configuration Checklist Description Format (XCCDF) – XCCDF provides a language for writing security checklists, benchmarks, and related documents
- Open Vulnerability and Assessment Language (OVAL) – OVAL provides a language for describing system information, including its current state, as well as the results of a vulnerability assessment.

Where OMG aims to improve tool interoperability through improved analysis of the output of multiple tools, NIST provides organizations with a common set of languages that can serve the basis for future interoperability efforts. While many of these standards may not be directly applicable in an embedded environment, they may be adapted for use. For example, XCCDF may be used to define checklists for embedded Linux distributions while OVAL can be used to describe the results of the assessment of an existing software intensive system. Taking advantage of existing NIST standards may be useful in ensuring that test results can be repeated and that mitigation strategies can be documented for the long period of time many embedded systems are deployed.

4.4.3 Security Analysis/Test Case/Test Scenario Definition

This sub-section focuses on a variety of techniques and methodologies for organizations interested in improving their security posture through improved testing definition. One major current area of research is in attack patterns. Attack patterns are primarily being investigated in government and industry and are commonly used to:

- Specify explicit requirements against which software security can be tested
- Avoid design and implementation issues that would make the software vulnerable to known attack patterns
- Include criteria and checks in design and code reviews, security tests, and post-deployment vulnerability assessments [10].

Attack patterns form the basis for abuse and misuse cases, threat models, and attack trees. However, because attack patterns rely on knowledge of attacks that have occurred in the past, they are a reactive technology. If attackers devise a new attack pattern, software developed and tested against attack patterns would remain vulnerable. Nevertheless, there is consensus that the improved discovery, avoidance, and mitigation of vulnerabilities to known attack patterns will improve the security of software in general. Taking advantage of other testing techniques in tandem with attack patterns will further improve software security.

Academia focuses primarily on attack trees, which are similar to attack patterns in many respects. Attack trees (also known as threat trees or attack graphs) provide a visual representation of possible attacks against the system. In practice, portions of an attack tree may be based on attack patterns as described earlier. Using this visual representation, organizations can better capture information for use in generating abuse and misuse cases, security requirements, and test patterns. In some environments, these attack trees may aid in the risk analysis process as well, aiding assessors in gaining a full understanding of the vulnerabilities within a system.

One of the primary drawbacks of attack trees in their current form is that it is difficult for individuals not skilled in security to effectively use them. Because a tree is generated based on security-relevant conditions within the software, unskilled analysts may generate incomplete trees. Similarly, analysts without a full grasp of the software system may make inadequate assumptions about the system itself.

5. Conclusions

Organizations wishing to take advantage of tools and techniques for software security testing need to be aware of the range of tools currently offered across the different types of tools and the specific aspects of the software development lifecycle against which these tools can be applied. The current software security market is focused almost entirely on source code analysis, resulting in a large number of robust tools that organizations may deploy to improve security analysis. In addition, organizations can take advantage of efforts under way to standardize reporting results among tools and evaluate these tools. Source code analysis tools are limited by the languages supported, primarily C and Java. Software written using other languages may have to rely on less robust or less well-understood source code analysis tools.

Due in large part to the increasing deployment of bytecode-based languages (e.g., Java and .NET), tools that perform bytecode analysis are increasingly prevalent. One of the primary drawbacks of these tools is that they are often coupled with source code analysis scanners. For example, tools that analyze Java or .NET-based code for security vulnerabilities perform bytecode analysis in tandem for improved understanding of the source code, resulting in fewer false positives and false negatives. Pure bytecode analysis tools still prove powerful and can be an asset to any organizations wishing to improve their software security posture.

The majority of current disassembler analysis tools specifically aid in the manual assessment of disassembled binary. Much of the existing research is aimed at improving the rate at which these tools can operate, automating much of the existing process. Fully decompiled source code, along with the original binary, allows organizations to take advantage of the robust technologies available in source code analysis tools against existing binary (either legacy or COTS) software.

Related to disassembly analysis tools is the field of binary analysis. As it currently stands, only one tool and service is available to perform binary analysis of existing executables. Organizations wishing to perform analysis of COTS products or legacy software can take advantage of Veracode’s technology. Outside of Veracode, there is little active academic research in the field of pure binary analysis, and much of this research focuses on disassembly to analyze binary components.

Both source-based and binary-based fault injection have a plethora of tools available on the market. Due to its status as one of the earliest research areas in non-functional testing and the heavy reliance on this form of testing used in the safety community, there is a good deal of research and commercially-available tools. Increasingly, fault injection tool vendors are also offering security-related services following the convergence between the software safety and security communities.

In the near future, organizations expect to be able to take advantage of additional tool categories. Specifically, pedigree analysis, which only recently became available with the popularity of open source software, will aid organizations in tracking source code components throughout the development lifecycle. Similarly, organizations should be able to use interoperability standards and methods currently being researched and deployed—either by OMG, NIST, or other organizations. With the ability to generate machine-readable reports from software security testing tools, organizations will be able to better analyze and compare the results and perform adequate risk analyses.

Over the long term, organizations will likely be able to take advantage of improvements in binary analysis, disassemble analysis, and malicious code detection—the research required to improve these types of tools is similar. Techniques for automating the understanding of binary code has the potential to drastically improve security testing capabilities throughout the software development lifecycle, much as techniques for automating understanding of source code has in the past few years.

6. References

[1] Michael Howard and Steve Lipner. "The Security Development Lifecycle." Microsoft Press, 2006.

[2] Steven Lavenhar. "Business Case" https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/code/212-BSI.html.

[3] Dennis Goldenson and Diane Gibson. "Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results." http://www.sei.cmu.edu/pub/documents/03.reports/pdf/03sr009-revised.pdf.

[4] Capers Jones. "Applied Software Measurement: Assuring Productivity and Quality." McGraw-Hill. New York, 1991.

[5] David H. Kitson and Stephen Masters. "An Analysis of SEI Software Process Assessment Results, 1987-1991." Proceedings of the Fifteenth International Conference on Software Engineering. Baltimore, Maryland. May 17 – 21, 1993. Washington, DC. IEEE Computer Society Press, 1993. Pages 68 – 77.

[6] Capers Jones. "Programming Productivity." McGraw-Hill. New York, 1986.

[7] Barry Boehm and Philip N. Papaccio. "Understanding and Controlling Software Costs." IEEE Transactions on Software Engineering 14, 10. October 1988. Pages 1462 – 1477.

[8] Jonathan Caulkins et al. "Optimizing Investments in Security Countermeasures." IEEE Security and Privacy. http://doi.ieeecomputersociety.org/10.1109/MSP.2007.117.

[9] Christoph Michael and Steven Lavenhar, Build Security In. "Source Code Analysis Tools – Overview." 2006. https://buildsecurityin.us-cert.gov/daisy/bsi/articles/tools/code/263-BSI.html.

[10] Karen Goertzel, et al. "Software Security Assurance." IATAC. http://iac.dtic.mil/iatac/download/security.pdf.

[11] Thomas Ball, Bell Laboratories. "The Concept of Dynamic Analysis." Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of Software Engineering, pages 216 – 234.

[12] IBM OLIVER. Wikipedia. http://en.wikipedia.org/wiki/IBM_OLIVER_(CICS_interactive_test/debug).

[13] SSLKeys. Debian Wiki. http://wiki.debian.org/SSLkeys.

[14] Paco Hope, Steven Lavenhar, Gunnar Peterson, and Cigital. "Architectural Risk Analysis." https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/architecture/10-BSI.html.

[15] R. Shirey. "Security Architecture for Internet Protocols: A Guide for Protocol Designs and Standards." Internet Draft, November 1994.

[16] Andrew Jaquith, Yankee Group, and CIO Asia. “A Few Good Metrics.” 2005. http://cio-asia.com/ShowPage.aspx?pagetype=2&articleid=2560&pubid=5&issueid=63.

[17] Zhichen Xu, Barton Miller, and Thomas Reps. “Safety Checking of Machine Code.” ACM SIGPLAN Notices. ftp://ftp.cs.wisc.edu/paradyn/papers/Xu00Safety.pdf.

[18] Louis Kruger. “The BREW Project at Sandia.” http://www.cs.wisc.edu/wisa/presentations/2005/02/louis.pdf.

[19] Karen Goertzel et al. “Enhancing the Development Life Cycle to Produce Secure Software.” DACS, 2008.

[20] Ari Takanen, Jared D. Demott, and Charles Miller. “Fuzzing for Software Security Testing and Quality Assurance.” Artech House, 2008.

[21] Peter Haggar, IBM developerWorks. “Java bytecode.” http://www.ibm.com/developerworks/ibm/library/it-haggar_bytecode/.

[22] Kris Britton (NSA CAS). “NSA Center for Assured Software.” Presentation to the Director of Central Intelligence’s Information Security and Privacy Advisory Board, Gaithersburg, MD. NIST Computer Security Division Computer Security Resource Center, March 21, 2006). *http://csrc.nist.gov/groups/SMA/ispab/documents/minutes/2006-03/K_Britton-March2006-color-ISPAB.pdf.*

APPENDIX A Tool Matrix

Product	AgileJ StructureViews
Description	Commercial Java visualization product that is deployed as an Eclipse Feature. The product brings together aspects of the Eclipse JDT Java Model, Set Theory, Class Diagrams, and XP/Agile Methods. The output resembles reverse engineered CASE tool drawings. The Eclipse JDT model performs a number of functions in the Eclipse Java IDE, including populating the package explorer and type hierarchy trees. AgileJ StructureViews taps into that same source of information to populate its class diagrams. The visualizations are UML class diagrams, which can be printed or exported as JPEG images. Class diagrams appear alongside the source file editor in the Eclipse IDE, and navigation is possible from any element on a diagram back to its source code. To comply with the XP goal of minimal documentation, no presentation-specific information is stored with a diagram. From the list of class names, all other information, including class members, inner classes, inheritances, associations and dependencies, is derived from Eclipse. The intention is that the diagrams only serve to increase comprehension of the coding model which they illustrate
URL	http://www.agilej.com/
Supported Languages	Java
Supported Platforms Where Tool Runs	Eclipse
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Risk Analysis
Lifecycle Position(s)	Design, Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	N/A
Ability of Testers to Modify Existing Rule Bases	N/A
Ability of Testers to Add New Rule Bases	N/A
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	N/A
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	antiparser
Description	A fuzz testing and fault injection API
URL	http://antiparser.sourceforge.net/
Supported Languages	N/A
Supported Platforms Where Tool Runs	
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Fuzz Testing
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	GPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Aspect Security AspectCheck
Description	Aspect Security uses AspectCheck in code review services to check for security-critical calls
URL	www.aspectsecurity.com/
Supported Languages	ASP.Net, C#, Java, VB.Net
Supported Platforms Where Tool Runs	
Supported Platform Where Target Resides	
Supported Compilers	
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Bytecode analysis
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	Unknown
Ability to Discover Debug Code	Unknown
Ability to Discover Unused Code	Unknown
Tool uses CWE Definitions of Vulnerabilities	Unknown
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Unknown
Ability of Testers to Add New Rule Bases	Unknown
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Unknown
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial Service
Licensing	
Vendor Technical Support	Unknown
Vendor Services / Professional services support	Yes
Required training or experience level to operate	Unknown
Vendor provided (or 3rd party provided) training available	Unknown
Comments	

Product	ASTRÉE
Description	Identifies undefined code constructs or run-time errors, such as out-of-bounds array indexing or arithmetic overflow.
URL	http://www.astree.ens.fr/
Supported Languages	C
Supported Platforms Where Tool Runs	Unix, Linux
Supported Platform Where Target Resides	
Supported Compilers	
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Advertises significant scalability
Ability to Identify Comments in Code	
Ability to Discover Debug Code	
Ability to Discover Unused Code	
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	N/A
Ability of Testers to Add New Rule Bases	N/A
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Research & Development Tool, Linux version is free
Licensing	BSD
Vendor Technical Support	None
Vendor Services / Professional services support	None
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	None
Comments	No updates being made to tool

Product	Black Duck Software
Description	Automatic tool for scanning source code and finding intellectual property and licensing issues. The analyses are performed based on patterns of licenses and known packages (e.g., SF.net). Unlike Palamida, Black Duck does not perform security-specific analyses. URL: http://www.blackducksoftware.com/index.html
URL	http://www.blackducksoftware.com/
Supported Languages	Unknown
Supported Platforms Where Tool Runs	Unknown
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	Yes
Finds or Checks for: (Tool Category)	Pedigree Analysis
Lifecycle Position(s)	Development, Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	1,000,000 LOC
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	**Boomerang**
Description	**open source decompiler**
URL	**http://boomerang.sourceforge.net/**
Supported Languages	C
Supported Platforms Where Tool Runs	
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Reverse Engineering
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Unknown
Ability of Testers to Add New Rule Bases	Unknown
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Unknown
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	BSD
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	BOON
Description	Performs integer range analysis to determine if an array can be indexed outside its bounds
URL	http://www.cs.berkeley.edu/~daw/boon/
Supported Languages	C
Supported Platforms Where Tool Runs	Unix, Linux
Supported Platform Where Target Resides	
Supported Compilers	GCC
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	N/A
Ability of Testers to Add New Rule Bases	N/A
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	
Vendor Technical Support	None
Vendor Services / Professional services support	None
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	None
Comments	No updates being made to tool

Product	Borland's Together
Description	A set of Eclipse plugins that provides UML 1.4 modeling, multi-language support, physical data modeling, design patterns, source code design pattern recognition, code template design and reuse, documentation generation, and code audits and metrics. Together adds language-neutral UML 2.0 diagramming, business process modeling, logical data modeling, and logical to physical data model transformation and custom pattern support. Borland's strategy is to focus on tool integration, enabling best-of-breed strategies. Borland's MDD strategy is to offer application development teams a choice of tools and approaches, rather than channeling them into a single modeling notation. Borland is active in the OMG, leading the Graphical Model Framework (GMF) for Eclipse, and is also active in the Open Systems Group. Borland's Together Visual modeling tool is based on UML2, BPMN, and MOF; it provides key features for software architects, designers, and coders. Together uses the OMG's QVT standard in model-to-model transformations and provides support for UML Object Constraint Language (OCL) 2.0. This includes audits and metrics, which are provided at both the model and code level and are defined in industry standard OCL. Overall, Borland provides a platform of common services and a set of practitioner tools within the broader scope of application life-cycle management
URL	http://www.borland.com/us/products/together/index.html
Supported Languages	UML, XML, QVT, OCL, MDA, EMF
Supported Platforms Where Tool Runs	Linux, Mac OS X, Solaris, Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Risk Analysis
Lifecycle Position(s)	Design, Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	N/A
Ability of Testers to Modify Existing Rule Bases	N/A
Ability of Testers to Add New Rule Bases	N/A
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	N/A
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No

Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	C Code Analyzer (CCA)
Description	Tests for out-of-bounds array indexing and arithmetic overflow.
URL	http://www.drugphish.ch/~jonny/cca.html
Supported Languages	C
Supported Platforms Where Tool Runs	Unix, Linux, Windows (cygwin)
Supported Platform Where Target Resides	Unix, Linux, Windows
Supported Compilers	GCC, MSVC
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Yes (successfully scanned the Linux kernel)
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	Yes (through modification)
Ability of Testers to Add New Rule Bases	Yes (through modification)
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	BSD
Vendor Technical Support	None
Vendor Services / Professional services support	None
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	None
Comments	The product is designed to minimize false positives.

Product	CenterLine Systems CodeCenter
Description	Detects incorrect pointer values, illegal array indices, bad function arguments, type mismatches, and uninitialized variables.
URL	http://www.ics.com/products/centerline/codecenter/
Supported Languages	C
Supported Platforms Where Tool Runs	Unix, Sun SPARC, Solaris
Supported Platform Where Target Resides	Unix
Supported Compilers	clcc, GCC, SPARC-C
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	N/A
Ability of Testers to Modify Existing Rule Bases	N/A
Ability of Testers to Add New Rule Bases	N/A
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	Commercial
Vendor Technical Support	Multiple tiers
Vendor Services / Professional services support	Yes
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	checKing
Description	Monitors the quality of software development process, including violations of coding rules.
URL	http://www.optimyth.com/checKing
Supported Languages	Java, JSP, JavaScript, XML, HTML
Supported Platforms Where Tool Runs	Unix, Linux
Supported Platform Where Target Resides	
Supported Compilers	Unknown
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	Yes
Ability to Discover Debug Code	Unknown
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	N/A
Ability of Testers to Modify Existing Rule Bases	Unknown
Ability of Testers to Add New Rule Bases	Unknown
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Unknown
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	Unknown
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Unknown
Comments	

Product	CodeScan Labs CodeScan
Description	Inspects web source code for security vulnerabilities and source code quality issues
URL	http://www.codescan.com/
Supported Languages	ASP
Supported Platforms Where Tool Runs	Windows
Supported Platform Where Target Resides	Windows
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	100,000 LOC
Ability to Identify Comments in Code	Yes
Ability to Discover Debug Code	Yes
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	"Regularly"
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	Annual Subscription Developer: $1000
Vendor Technical Support	Technical support and software maintenance are included as part of the subscription contract.
Vendor Services / Professional services support	
Required training or experience level to operate	
Vendor provided (or 3rd party provided) training available	
Comments	

Product	Compuware DevPartner SecurityChecker
Description	Identifies known and potential security vulnerabilities
URL	http://www.compuware.com
Supported Languages	C
Supported Platforms Where Tool Runs	Windows
Supported Platform Where Target Resides	Windows
Supported Compilers	MSVC, Java JDK
Can Tool be used Remotely?	Yes
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	Yes
Ability to Discover Debug Code	Yes
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Unknown
Ability of Testers to Add New Rule Bases	Unknown
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	Free, Professional
Vendor Technical Support	Yes
Vendor Services / Professional services support	Yes
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	Coverity Prevent
Description	Finds quality problems and security vulnerabilities using static source code analysis. Checks include: -API usage errors -Buffer overflow -Buffer overflows -Cross-site scripting -Dangling stack references -Denial of service -File corruption -Flawed branch logic -Format string vulnerabilities -Improper bounds checking -Incorrect allocation sizes -Insecure access control -Integer overflows -Logic errors -Memory corruption -Memory leaks -Non-null terminated strings -Null pointer dereferences -Out-of-bounds array access -Privilege escalations -SQL injection -Stack overflow -Stack smashing -Stack string overruns -System resource leaks -Use of freed resources -Use of uninitialized data
URL	http://www.coverity.com
Supported Languages	C, C++, C#, Java
Supported Platforms Where Tool Runs	FreeBSD, HP-UX, Linux, Mac OS X, NetBSD, Solaris SPARC, Solaris X86, Windows
Supported Platform Where Target Resides	
Supported Compilers	Arm CC, G++, GCC, Green Hills compiler, HP-UX compiler, IAR compiler, Intel compiler for C/C++, Intel Microsignal Architecture compiler, Java JDK (1.4 and higher), Metrowerks CodeWarrior, MS Visual Studio, PICC compiler, Sun CC, TI Code Composer C compiler, Wind River Diab compiler
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Up to 4,000,000 LOC
Ability to Identify Comments in Code	Yes
Ability to Discover Debug Code	Yes
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	Unknown
Frequency of Rule Base Updates by Tool Provider	Unknown

Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	Commercial
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	CppCheck
Description	Identifies memory leaks, buffer overruns and many other common errors
URL	http://cppcheck.wiki.sourceforge.net/
Supported Languages	C
Supported Platforms Where Tool Runs	Linux, Windows
Supported Platform Where Target Resides	
Supported Compilers	GCC, DJGPP
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Monthly
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Open Source
Licensing	GPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	CQual
Description	Uses type qualifiers to perform a taint analysis, which detects format string vulnerabilities
URL	http://sourceforge.net/projects/cqual/
Supported Languages	C
Supported Platforms Where Tool Runs	Unix, Linux
Supported Platform Where Target Resides	
Supported Compilers	GCC
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Open Source
Licensing	GPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Csur
Description	Identifies cryptographic protocol-related vulnerabilities
URL	http://www.lsv.ens-cachan.fr/Software/csur/
Supported Languages	C
Supported Platforms Where Tool Runs	Unknown
Supported Platform Where Target Resides	Unknown
Supported Compilers	Unknown
Can Tool be used Remotely?	Unknown
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Unknown
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	Unknown
Ability to Discover Debug Code	Unknown
Ability to Discover Unused Code	Unknown
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	Unknown
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Dfuz
Description	A remote protocol fuzzer
URL	http://genexx.org/dfuz/
Supported Languages	N/A
Supported Platforms Where Tool Runs	Linux, Unix
Supported Platform Where Target Resides	Linux, Unix
Supported Compilers	N/A
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Fuzz Testing
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	"Regularly"
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	Limited
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Eau Claire
Description	Identifies array bounds errors, null pointer dereferences, and the use of dangerous string functions
URL	http://www.vantuyl.com/chess/EauClaire/
Supported Languages	C
Supported Platforms Where Tool Runs	Unix, Windows
Supported Platform Where Target Resides	
Supported Compilers	Unknown
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	By request
Licensing	Unknown
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Unknown
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	EULAyzer
Description	Open source tool for analyzing license agreements to find interesting words and phrases. In evaluation of the latest version (June 2006), the tool did not perform well and conducted no security-relevant analyses.
URL	http://eulayzer.en.softonic.com/
Supported Languages	N/A
Supported Platforms Where Tool Runs	Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Pedigree Analysis
Lifecycle Position(s)	Acquisition
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	Yes
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Exhaustif
Description	A binary fault injection tool that aims to simulate many of the errors that may occur while software is deployed. Specifically aimed at embedded systems and control systems, which must have high fault tolerance, Exhaustif has been extended to support distributed systems.
URL	http://www.exhaustif.es/
Supported Languages	N/A
Supported Platforms Where Tool Runs	RTEMS
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	Yes
Finds or Checks for: (Tool Category)	Fault Injection
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Unknown
Ability of Testers to Add New Rule Bases	Unknown
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Unknown
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Unknown
Vendor Services / Professional services support	Unknown
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	Unknown
Comments	

Product	Fakebust
Description	A program that controls executable activity to assess potentially malicious programs
URL	http://freshmeat.net/projects/fakebust/
Supported Languages	C
Supported Platforms Where Tool Runs	Linux
Supported Platform Where Target Resides	Linux
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Malicious Code Detector
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	Unknown
Ability to Discover Debug Code	Unknown
Ability to Discover Unused Code	Unknown
Tool uses CWE Definitions of Vulnerabilities	Unknown
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Unknown
Ability of Testers to Add New Rule Bases	Unknown
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Unknown
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	GPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	FindBugs
Description	Static bytecode analyzer based on Jakarta BCEL
URL	http://findbugs.sourceforge.net/
Supported Languages	Java
Supported Platforms Where Tool Runs	Unix, Linux
Supported Platform Where Target Resides	
Supported Compilers	Sun JDK, Eclipse, Netbeans, Jboss
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Twice-monthly
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Open Source
Licensing	LGPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Flawfinder
Description	Detects use of risky functions, buffer overflow (strcpy()), format string ([v][f]printf()), race conditions (access(), chown(), and mktemp()), shell metacharacters (exec()), and poor random numbers (random()).
URL	http://www.dwheeler.com/flawfinder/
Supported Languages	C
Supported Platforms Where Tool Runs	Unix, Linux
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Open Source
Licensing	GPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Fluid
Description	Analysis based verification for attributes such as race conditions, thread policy, and object access with no false negatives. Emphasis on code safety, security, API compliance, and other attributes of dependability.
URL	http://www.fluid.cs.cmu.edu:8080/Fluid
Supported Languages	Java
Supported Platforms Where Tool Runs	Windows, Linux
Supported Platform Where Target Resides	
Supported Compilers	Eclipse
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Not available
Licensing	
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Fortify Source Code Analysis Suite 5.2
Description	Fortify's analysis is done at a semantic, rather than syntactical, level. For example, it can map out data flows and recognize that untested, user-entered data -- always a potential threat -- has been passed to a routine. Because the Fortify engine understands the code, it can monitor execution and data flows through multiple modules and identify the points where unsafe data is touched without first being verified. Few solutions today can find intermodule security problems of this kind. Fortify checks over 118 vulnerability categories including : -Buffer Overflow -Command Injection -Cross-Site Scripting -Denial of Service -Format String -Integer Overflow -Log Forging -Password Management -Path Manipulation -Privacy Violation -Race Condition Fortify generates a large XML file containing data on all the vulnerabilities it finds. This file is then analyzed by the Workbench, which displays the information in a user-friendly format. Unless programmers are up-to-date on the nature of specific coding vulnerabilities, they are likely to be surprised by what Fortify flags. The product catches not only buffer over-runs and opportunities for SQL injection, but also more-esoteric issues. Because the number of generated warnings can be rather large, the Audit Workbench automatically assigns them severity ratings and enables the creation of filters, so that only items of interest are displayed. The display not only lists the vulnerabilities and the explanations, but also takes developers directly to the offending line of code. Fortify SCA results can integrate with Fortify Software's centralized, Web-based reporting and control console to make findings and metrics, including trends within projects, comparisons between groups and more, available to key stakeholders
URL	http://www.fortify.com
Supported Languages	ASP.NET, Classical .ASP, C, C++, C#, Java, JSP, PHP, PL/SQL, T-SQL, VB.NET, Visual Basic 6, XML, COBOL
Supported Platforms Where Tool Runs	Windows, Solaris, Linux, HP-UX, AIX and Mac OS X
Supported Platform Where Target Resides	
Supported Compilers	IBM Rational Application Developer, Eclipse, MS .NET, GCC
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development

Scalability (Ability to scan up to 1,000,000 LOC?)	1,000,000 LOC
Ability to Identify Comments in Code	Yes
Ability to Discover Debug Code	Yes
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Quarterly
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	Yes
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	Grammatech CodeSonar
Description	Identifies null-pointer dereferences, divide-by-zeros, buffer over- and underruns
URL	www.grammatech.com/products/codesonar/
Supported Languages	C, C++, Ada
Supported Platforms Where Tool Runs	Windows, Linux, Solaris, OS X
Supported Platform Where Target Resides	
Supported Compilers	ARM, GCC, Green Hills C Compiler, Hi-Tech C Compiler, Intel C/C++ Compiler, MSVS, Renesas, Sun CC, TiCodeComposer, Wind River
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Green Hills Software DoubleCheck
Description	Identifies like buffer overflows, resource leaks, invalid pointer references, and violations of ... MISRA.
URL	www.ghs.com/products/doublecheck.html
Supported Languages	C, C++
Supported Platforms Where Tool Runs	Supports embedded development with the following processors: Power Architecture, x86/Pentium, Blackfin, MIPS, Intrinsity, Lexra, StarCore, M32R, ColdFire, 680x0/683xx, CPU32, SH, Alpha, ZSP, ARM/Thumb, XScale, StrongARM, SPARC/SPARClite, V8xx, MCORE, TriCore, ST100, i960, RAD6000, RH32, FR
Supported Platform Where Target Resides	Embedded systems
Supported Compilers	Green Hills, GCC, Arm CC, Hi-Tech C Compiler, Intel C/C++ Compiler, MSVC, Renesas, Sun CC, Ti CodeComposer, Wind River C and C++
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	Grid-FIT
Description	An implementation of the FIT framework that performs network-level fault injection on deployments of the Globus Grid and other Web services-based systems
URL	http://www.allhands.org.uk/2006/proceedings/papers/607.pdf
Supported Languages	SOAP
Supported Platforms Where Tool Runs	Globus Grid
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	Yes
Finds or Checks for: (Tool Category)	Fault Injection
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Unknown
Ability of Testers to Add New Rule Bases	Unknown
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Unknown
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Research
Licensing	
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	GPF
Description	Automated testing technique (fuzzing) to find bugs in software
URL	http://www.appliedsec.com/developers.html
Supported Languages	N/A
Supported Platforms Where Tool Runs	Linux, Unix
Supported Platform Where Target Resides	Linux, Unix
Supported Compilers	N/A
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Fuzz Testing
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	GPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Hammurapi
Description	Has a number of inspectors which catch potential programmer errors.
URL	http://www.hammurapi.biz/hammurapi-biz/ef/xmenu/hammurapi-group/index.html
Supported Languages	Java
Supported Platforms Where Tool Runs	Linux, Unix, Windows
Supported Platform Where Target Resides	
Supported Compilers	Sun JDK, Eclipse
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	1,000,000 LOC
Ability to Identify Comments in Code	Yes (if configured)
Ability to Discover Debug Code	Unknown
Ability to Discover Unused Code	Unknown
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Open Source
Licensing	Apache 2.0, LGPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Hex-Rays (formerly DataRescue) IDAPro
Description	Static, interactive disassembler that targets Windows and Linux, .NET, JVM; and multi-processor (most CPUs, including Intel, and RISC) assembler languages.
URL	http://www.hex-rays.com/idapro/
Supported Languages	N/A
Supported Platforms Where Tool Runs	JVM, Linux, .Net, Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Disassembler Analysis
Lifecycle Position(s)	Testing, Acquisition
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	$775-$1470
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Holodeck
Description	A binary fault injection tool with security-specific capabilities. Through a mixture of fault injection and fuzz testing, Holodeck aims to provide improved security testing through its dynamic analysis capabilities.
URL	http://www.securityinnovation.com/holodeck/
Supported Languages	N/A
Supported Platforms Where Tool Runs	Windows
Supported Platform Where Target Resides	Windows
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Fault Injection
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	Yes
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	IBM's Rational Software Architect (RSA) Rebranded for WebSphere
Description	A comprehensive modeling and development environment that leverages the UML for designing architecture for C++ and Java 2 Enterprise Edition (J2EE) applications and web services. RSA is built on the Eclipse open-source software framework and includes capabilities focused on architectural code analysis, C++, and MDD with the UML for creating resilient applications and web services. RSA provides visual construction tools to expedite software design and development and supports reverse transformations from Java to UML, and C++ to UML. RSA enables model management for architectural re-factoring (e.g., split, combine, compare and merge models and model fragments)
URL	http://www-01.ibm.com/software/awdtools/swarchitect/websphere/
Supported Languages	Java, UML
Supported Platforms Where Tool Runs	Linux, Windows
Supported Platform Where Target Resides	
Supported Compilers	Eclipse
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Risk Analysis
Lifecycle Position(s)	Design, Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	N/A
Ability of Testers to Modify Existing Rule Bases	N/A
Ability of Testers to Add New Rule Bases	N/A
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	N/A
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	ITS4
Description	Detects potentially dangerous function calls, Provides simple risk analysis
URL	www.cigital.com/its4/
Supported Languages	C
Supported Platforms Where Tool Runs	Linux, Unix, Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	!,000,000 LOC
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	ITS4 License
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Jlint
Description	Finds bugs, inconsistencies and synchronization problems
URL	http://jlint.sourceforge.net
Supported Languages	Java
Supported Platforms Where Tool Runs	Linux, Unix, Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Open Source
Licensing	GPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Klocwork K7
Description	Klocwork's patented static source code analysis technology implements management Insight, Auditor Analysis, and Developer Assistance across the following critical development challenges. · Finds security vulnerabilities in software and improve overall application security. · Understands large code bases and simplify their structure. · Measures and tracks key quality indicators throughout the release cycle. · Extensible, allowing customization of the analysis to suit an organization's quality and security priorities. C/C++ Vulnerability Categories · Access problems · Buffer overflow · DNS spoofing · Ignored return values · Injection flaws · Insecure storage · Unvalidated user input Java Vulnerability Categories · Denial of Service · Injection Flaws (e.g. SQL Injection) · Unvalidated Input · Mobile Code Security · Broken Session Management · Cross-Site Scripting · Improper Errors Handling · Broken Access Control K7 can perform analysis based on Java source code and bytecodes, the latter being Java's form of executable file. If the bytecodes contain debug information, K7 can trace defects back to specific lines of code. If not, it can simply identify that a certain type of bug has been found. This option enables sites that rely on third-party Java components to screen them for possible defects before use and to identify the type of defect to the vendor. A separate utility presents a detailed pictorial analysis of the complex relationships between files and functions. It can identify odd relationships that would indicate bugs, such as a library of functions making calls to an application. K7 also has extensive reporting capabilities. The management console can generate an extensive PDF file (filters enable managers to include or exclude a wide variety of data), exportable text, or XML files.
URL	http://www.klocwork.com/
Supported Languages	C, C++, Java, Supported IDEs, Eclipse 3.0, 3.1, 3.2, Microsoft Visual Studio 6, 2002, 2003, IBM Rational Application Developer 6.0, Wind River Workbench 2.3, 2.4, 2.5, QNX Momentics 6.3 (SP2)
Supported Platforms Where Tool Runs	Solaris 8, Solaris 9, Solaris 10, Red Hat Linux 9.0, Red Hat Enterprise Linux 3.0, Red Had Enterprise Linux 4.0, SUSE Linux 9.3, Windows XP Professional, Windows 2000 Professional, Windows 2000 Server, Windows Server 2003
Supported Platform Where Target Resides	
Supported Compilers	GNU GCC/G++, ARM, Microsoft Visual C++, Green Hills, Wind River Diab, Sun Forte, MetroWerks, Metaware, Hitachi h38, TI tms470, Sun Java Compiler
Can Tool be used Remotely?	

Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	Yes
Ability to Discover Debug Code	Yes
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	Yes
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	LAPSE
Description	Helps audit Java J2EE applications for common types of security vulnerabilities found in Web applications.
URL	http://suif.stanford.edu/~livshits/work/lapse/
Supported Languages	Java
Supported Platforms Where Tool Runs	Linux, Unix
Supported Platform Where Target Resides	
Supported Compilers	Eclipse
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	GPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	No
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	LDRA Software Technology TBSecure Plugin
Description	The TBsecure plug-in to TBvision comes complete with the Carnegie Mellon Software Engineering Institute (SEI) CERT C secure coding standard. TBsecure identifies security vulnerabilities and enables implementation of the recent CERT C Secure Coding Standard version 1.0.
URL	www.ldra.com/
Supported Languages	C
Supported Platforms Where Tool Runs	Supports the following processors for embedded development: ARM7/ARM9, Freescale MC68K, MPC5xx, 6xx & 8xx, Infineon TriCore, C166, Intel 8051, 80C196, MIPS SmartMIPS, 4Kx, PowerPC, 5xx, 6xx, 7xx & 8xx, Renesas Super H xx, TI TMS320Cxx
Supported Platform Where Target Resides	
Supported Compilers	Unknown
Can Tool be used Remotely?	Yes
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	Unknown
Ability to Discover Unused Code	Unknown
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	MagicDraw
Description	Visual UML modeling and CASE tool. This tool facilitates analysis and design of object-oriented systems and databases. MagicDraw provides full round-trip support for J2EE, C#, C++, CORBA IDL programming languages, .NET, XML Schema, and WSDL), as well as database schema modeling, DDL generation, and reverse engineering facilities. MagicDraw's reverse engineering capability allows generation of UML models from Java, C#, C++, CORBA IDL, EJB 2.0, DDL, CIL (MSIL), WSDL, and XML Schema source code. In addition, the product supports automatic generation of sequence diagrams from Java source code and adds a more detailed view of the system. MagicDraw's automatic report generation engine produces comprehensive requirements, software design documentation, and other types of reports in HTML, PDF, and RTF formats. MagicDraw UML generates standard artifacts that match industry standard software development processes. The report engine allows users to generate up-to-date reports based on your own templates with layout and formatting specified. MagicDraw runs on a wide variety of operating systems, including Windows 98/ME/NT/2000/XP/Vista, Solaris, OS/2, Linux, HP-UX, AIX, MacOS (X), and other platforms that support Java 5 and 6
URL	http://www.magicdraw.com/
Supported Languages	Java, C++, UML, IDL
Supported Platforms Where Tool Runs	Linux, Mac OS X, Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Risk Analysis
Lifecycle Position(s)	Design, Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	N/A
Ability of Testers to Modify Existing Rule Bases	N/A
Ability of Testers to Add New Rule Bases	N/A
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	N/A
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	$149-$2460
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	High

Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	MEFISTO
Description	The Multi-level Error/Fault Injection in Simulation Tool (MEFISTO) is a source-code fault injection tool for the VHDL hardware description language. Developed in the mid 1990s, MEFISTO is one of the most commonly referenced tools for performing source-based fault injection
URL	http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?isnumber=7613&arnumber=315656&count=48&index=40
Supported Languages	VHDL
Supported Platforms Where Tool Runs	Unknown
Supported Platform Where Target Resides	
Supported Compilers	Unknown
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Fault Injection
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Unknown
Ability of Testers to Add New Rule Bases	Unknown
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Unknown
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Research
Licensing	
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	MicroGOLD's WithClass
Description	A feature-rich UML modeling tool. This product allows the user to draw UML diagrams, generate code, and reverse engineer popular OO languages, including C++. Java, Delphi , VB, IDL, Perl, PHP, C#, and VB.Net. WithClass draws all UML type 1.x diagrams. VBA can be used to create add-ins to extend the functionality of the tool
URL	http://www.microgold.com/version3/products.html
Supported Languages	C, C++, Java, Delphi, VB, IDL, Perl, PHP, C#, VB.net, UML
Supported Platforms Where Tool Runs	Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Risk Analysis
Lifecycle Position(s)	Design, Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	N/A
Ability of Testers to Modify Existing Rule Bases	N/A
Ability of Testers to Add New Rule Bases	N/A
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	N/A
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Microsoft fxCop
Description	Performs static analysis for Microsoft .NET programs that compile to CIL
URL	http://msdn.microsoft.com/en-us/library/bb429476(vs.80).aspx
Supported Languages	C#
Supported Platforms Where Tool Runs	Windows
Supported Platform Where Target Resides	
Supported Compilers	MSVS
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	1,000,000 LOC
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free standalone version and integrated in some Microsoft Visual Studio editions
Licensing	
Vendor Technical Support	
Vendor Services / Professional services support	
Required training or experience level to operate	
Vendor provided (or 3rd party provided) training available	
Comments	

Product	Microsoft PREfix and PREfast
Description	Identifies known and potential security vulnerabilities
URL	https://research.microsoft.com/pubs/69125/icse05exp.pdf
Supported Languages	C
Supported Platforms Where Tool Runs	Windows
Supported Platform Where Target Resides	
Supported Compilers	MSVS
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Integrated in some Microsoft Visual Studio editions
Licensing	
Vendor Technical Support	
Vendor Services / Professional services support	
Required training or experience level to operate	
Vendor provided (or 3rd party provided) training available	
Comments	

Product	M Squared Technologies Resource Standard Metrics (RSM)
Description	Scans for 50 readability or portability problems or questionable constructs, e.g. different number of "new" and "delete" key words or an assignment operator (=) in a conditional (if).
URL	http://msquaredtechnologies.com/m2rsm
Supported Languages	C, C++, C#, Java
Supported Platforms Where Tool Runs	Linux, Unix, Windows
Supported Platform Where Target Resides	
Supported Compilers	MS .NET, Eclipse, Jbuilder, Sun JDK
Can Tool be used Remotely?	Yes
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	Yes
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	ObjectWeb ASM
Description	Performs Java bytecode manipulation and analysis framework. This tool can be used to modify existing Java classes or to dynamically generate classes directly in binary form. The framework includes common transformations and analysis algorithms to enable easy assembly of custom-defined complex transformations and to custom-generate code analysis tools. This tool is being used by several research projects interested in safety or security analysis of Java bytecode.
URL	http://asm.objectweb.org/
Supported Languages	Java
Supported Platforms Where Tool Runs	Java
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Bytecode analysis
Lifecycle Position(s)	Testing, Acquisition
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	Custom
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	OPTIMA Business Information Technology GmbH OPTIMA Bytecode Scanner
Description	This tool is used in OPTIMA's code review services. The tool performs pattern and data flow analyses, and is designed to detect the maximum number of security problems even at the risk of generating more false positives. For example, the tool can be modified to perform a validation that "untaints" an object. OPTIMA has used this tool to analyze cryptography and other security-specific functional implementations. The tool is scalable for large projects (large code bases) and has been used by OPTIMA on a number of such code reviews. The tool is not fully automated. OPTIMA relies on expert human analysis assisted by the tool.
URL	http://www.optimabit.com/en/optima/home.html
Supported Languages	Java
Supported Platforms Where Tool Runs	Java
Supported Platform Where Target Resides	
Supported Compilers	
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Bytecode analysis
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	Unknown
Ability to Discover Debug Code	Unknown
Ability to Discover Unused Code	Unknown
Tool uses CWE Definitions of Vulnerabilities	Unknown
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Unknown
Ability of Testers to Add New Rule Bases	Unknown
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Unknown
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial Service
Licensing	
Vendor Technical Support	Unknown
Vendor Services / Professional services support	Yes
Required training or experience level to operate	Unknown
Vendor provided (or 3rd party provided) training available	Unknown
Comments	

Product	Ounce 4.0
Description	Automatically analyzes source code through the use of a language processor, which parses the application to create a Common Intermediate Security Language (CISL). The CISL captures multi-dimensional information about each call site, allowing Ounce to refine vulnerability data through three different levels of analysis. Ounce determines vulnerabilities by tracking the flow of data through an application, using cross-module, cross-language, semantic and data flow analysis to understand the complex interrelationships between individual calls, modules, data elements, and processes. Ounce separates real vulnerabilities from potential ones, allowing security analysts, QA teams, and developers to click instantly to confirmed vulnerabilities for focused remediation efforts. Ounce additionally sorts results by severity (high, medium, low) as well as by type (buffer overflow, race condition, privilege escalation, etc.), and Ounce's Security Knowledgebase offers suggestions to the developer for correcting the vulnerability or exception. Ounce allows the developer to make the choice to correct or modify the code on a case by case basis as the developer typically understands more about the desired behavior of the application. Customers may tailor the Security Knowledgebase to specific security and policy standards, and apply those standards consistently across the enterprise. Vulnerability Categories · Buffer overflows · Privilege escalation · Race conditions in C and C++ · Input validation errors · SQL injection vulnerabilities · Cross site scripting errors · Basic design Flaws (such as proper implementation of access control) · Basic Policy Violations (such as is cryptography in use and is it strong enough)
URL	http://www.ouncelabs.com/solutions/ounce-source-code-analysis.asp
Supported Languages	C, C++, Java, JSP, ASP.NET, VB.NET, C#, Supported IDEs, Microsoft Visual Studio, IBM Rational Application Developer, Eclipse
Supported Platforms Where Tool Runs	AIX, Linux, Solaris. Windows
Supported Platform Where Target Resides	
Supported Compilers	MS VC, Eclipse, Sun JDK
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	
Ability to Identify Comments in Code	Yes
Ability to Discover Debug Code	Yes
Ability to Discover Unused Code	Yes

Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	Package javassist.bytecode.analysis (Javassist API)
Description	The Java Programming Assistant (Javassist) tool includes an API for performing data-flow analysis on a Java method's bytecode, enabling the analyst to determine the type state of the stack and local variable table at the beginning of each instruction. This API can also be used to validate bytecode, find dead bytecode, and identify unnecessary checkcasts.
URL	http://www.csg.is.titech.ac.jp/~chiba/javassist/html/javassist/bytecode/analysis/package-summary.html and http://www.csg.is.titech.ac.jp/~chiba/javassist/
Supported Languages	Java
Supported Platforms Where Tool Runs	N/A
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Bytecode analysis
Lifecycle Position(s)	Testing, Acquisition
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	Apache
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Palamida
Description	Automates scanning source code and finding intellectual property and licensing issues. Palamida offers additional services aimed towards resolving security defects occurring in externally developed software.
URL	http://www.palamida.com
Supported Languages	Java, JavaScript, C#, C, C++, Perl, Python, PHP, VB
Supported Platforms Where Tool Runs	Linux, Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	Yes
Finds or Checks for: (Tool Category)	Pedigree Analysis
Lifecycle Position(s)	Development, Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	1,000,000 LOC
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	Parasoft C++Test
Description	Identifies defects, poor constructs, potentially malicious code and other elements
URL	http://www.parasoft.com/jsp/products/home.jsp?product=CppTest
Supported Languages	C++
Supported Platforms Where Tool Runs	Linux, Mac OS X, Solaris, Windows
Supported Platform Where Target Resides	
Supported Compilers	GCC, MSVS, MingW, Sun C++
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	Parasoft Jtest
Description	Identifies defects, poor constructs, potentially malicious code and other elements
URL	http://www.parasoft.com/jsp/products/home.jsp?product=Jtest
Supported Languages	Java
Supported Platforms Where Tool Runs	Linux, Mac OS X, Solaris, Windows
Supported Platform Where Target Resides	
Supported Compilers	Sun JDK
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	Parasoft .TEST
Description	Identifies defects, poor constructs, potentially malicious code and other elements
URL	http://www.parasoft.com/jsp/products/home.jsp?product=TestNet
Supported Languages	C#, VB.Net, MC++
Supported Platforms Where Tool Runs	Linux, Mac OS X, Solaris, Windows
Supported Platform Where Target Resides	
Supported Compilers	MSVS
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	Peach Fuzzing Platform
Description	A SmartFuzzer that is capable of performing both generation and mutation based fuzzing.
URL	http://peachfuzzer.com/
Supported Languages	N/A
Supported Platforms Where Tool Runs	Linux, Mac OS X, Unix, Windows
Supported Platform Where Target Resides	Linux, Mac OS X, Unix, Windows
Supported Compilers	N/A
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Fuzz Testing
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	MIT
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	PMD
Description	Identifies questionable constructs, dead code, duplicate code
URL	http://pmd.sourceforge.net/
Supported Languages	Java
Supported Platforms Where Tool Runs	Linux, Unix
Supported Platform Where Target Resides	
Supported Compilers	Sun JDK
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Open Source
Licensing	BSD
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Praxis SPARK tool set
Description	Identifies ambiguous constructs, data- and information-flow errors, any property expressible in first-order logic (Examiner, Simplifier, and SPADE)
URL	http://www.praxis-his.com/sparkada/pdfs/SPARK_Brochure.pdf
Supported Languages	SPARK (Ada subset)
Supported Platforms Where Tool Runs	Linux, Mac OS X, Open VMS, Solaris, Windows
Supported Platform Where Target Resides	
Supported Compilers	SPARK
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	Yes
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	Programming Research QA
Description	Detects out-of-bounds array indexing
URL	http://www.programmingresearch.com/QAC_MAIN.html
Supported Languages	C, C++, Fortran, Java
Supported Platforms Where Tool Runs	Unix, Windows
Supported Platform Where Target Resides	
Supported Compilers	Extensive
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	Yes
Ability to Discover Debug Code	Unknown
Ability to Discover Unused Code	Unknown
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	
Vendor Services / Professional services support	
Required training or experience level to operate	
Vendor provided (or 3rd party provided) training available	
Comments	

Product	RATS (Rough Auditing Tool for Security)
Description	Identifies potential security risks
URL	http://www.securesoftware.com/resources/tools.html
Supported Languages	C, Perl, Python, PHP
Supported Platforms Where Tool Runs	Linux, Unix, Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Open Source
Licensing	GPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	ReSharper
Description	Add-on for Visual Studio which provides static code analysis for C#.
URL	http://www.jetbrains.com/resharper/
Supported Languages	C#
Supported Platforms Where Tool Runs	Windows
Supported Platform Where Target Resides	
Supported Compilers	MSVS
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Active)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Simics Hindsight
Description	Reverse execution debugger by virtutech
URL	http://www.virtutech.com/products/simics_hindsight.html
Supported Languages	
Supported Platforms Where Tool Runs	Eclipse plugin
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Reverse Engineering
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Unknown
Ability of Testers to Add New Rule Bases	Unknown
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Unknown
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Unknown
Licensing	Commercial
Vendor Technical Support	Unknown
Vendor Services / Professional services support	Unknown
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Unknown
Comments	

Product	Smatch
Description	Simple scripts look for problems in simplified representation of code. primarily for Linux kernel code
URL	http://smatch.sourceforge.net/
Supported Languages	C
Supported Platforms Where Tool Runs	Linux
Supported Platform Where Target Resides	
Supported Compilers	GCC
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	1,000,000 LOC
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Open Source
Licensing	GPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	SoftCheck Inspector
Description	Creates assertions for each module, tries to prove the system obeys assertions and the absence of runtime errors. Statically determines and documents pre- and postconditions for Java methods. Statically checks preconditions at all call sites
URL	http://www.sofcheck.com/products/inspector.html
Supported Languages	Ada, Java
Supported Platforms Where Tool Runs	Windows
Supported Platform Where Target Resides	
Supported Compilers	AdaMagic, AppletMagic
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Unknown
Vendor Services / Professional services support	Unknown
Required training or experience level to operate	Unknown
Vendor provided (or 3rd party provided) training available	Unknown
Comments	

Product	Sparx System's Enterprise Architect
Description	A comprehensive UML analysis and design tool. Enterprise Architect provides complete traceability from requirements analysis and design artifacts through to implementation and deployment. Enterprise Architect is built upon the UML 2 specification. This product can display UML profiles to extend the modeling domain and includes model validation to ensure integrity. This tool combines business processes, information, and work flows into one model using extensions for BPMN and the Eriksson-Penker profile. Supported diagrams include object, composite, package, component, deployment, use case, communication, sequence, interaction, activity, state, and timing
URL	http://www.sparxsystems.com.au/products/ea/index.html
Supported Languages	UML, C++, Java, C#, VB.net, VB, Delphi, PHP, Python, ActionScript
Supported Platforms Where Tool Runs	Linux, Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Risk Analysis
Lifecycle Position(s)	Design, Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	N/A
Ability of Testers to Modify Existing Rule Bases	N/A
Ability of Testers to Add New Rule Bases	N/A
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	N/A
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	SPIKE
Description	A fuzzer that exposes APIs for quickly and efficiently developing network fuzzer protocols
URL	http://www.immunitysec.com/resources-freesoftware.shtml
Supported Languages	N/A
Supported Platforms Where Tool Runs	Linux
Supported Platform Where Target Resides	Linux
Supported Compilers	N/A
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Fuzz Testing
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	"Regularly"
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	GPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	No
Comments	Extensive Knowledge of C is required

Product	Splint
Description	Identifies security vulnerabilities and coding mistakes. Similar to Lint.
URL	http://www.splint.org/
Supported Languages	C
Supported Platforms Where Tool Runs	Linux, Solaris, Unix, Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	1,000,000 LOC
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Open Source
Licensing	GPL
Vendor Technical Support	
Vendor Services / Professional services support	
Required training or experience level to operate	
Vendor provided (or 3rd party provided) training available	
Comments	

Product	StackFrame, LLCTorqueWrench
Description	This static Java bytecode analysis tool looks for violations of coding standards and practices and some coding problems. This tool is probably not relevant for the applications being reviewed for this paper.
URL	http://www.stackframe.com/TorqueWrench/
Supported Languages	Java
Supported Platforms Where Tool Runs	Java
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Bytecode analysis
Lifecycle Position(s)	Testing, Acquisition
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Sulley
Description	A fuzzer development and fuzz testing framework consisting of multiple extensible components.
URL	http://code.google.com/p/sulley/
Supported Languages	N/A
Supported Platforms Where Tool Runs	
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	
Finds or Checks for: (Tool Category)	Fuzz Testing
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	GPL
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Sun Microsystems JFluid
Description	This tool requires the instrumentation of the bytecode or object code that will be analyzed by the tool. This tool may still be considered an R&D application and is not yet ready for extensive use.
URL	http://research.sun.com/techrep/2003/smli_tr-2003-125.pdf
Supported Languages	Java
Supported Platforms Where Tool Runs	Java
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Bytecode analysis
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	In Research
Licensing	
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Swat4j
Description	A model based, goal oriented source code auditing tool
URL	http://www.codeswat.com/
Supported Languages	Java
Supported Platforms Where Tool Runs	Eclipse plugin
Supported Platform Where Target Resides	
Supported Compilers	Eclipse
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Development
Scalability (Ability to scan up to 1,000,000 LOC?)	1,000,000 LOC
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	Yes
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Trusted Labs TL ADT
Description	This tool is used as part of Trusted Labs' security evaluation services (including white box (source code) static analysis). The Trusted Labs analyst runs TL ADT to automate code reviews in connection with C&A using TL ADT. The tool is used to challenge the Java application bytecode (in a CAP file) against a set of security rules defined by or for the certification authority. Implementing the rules inside TL ADT allows the Trusted Labs analyst to examine the source code.
URL	http://www.trusted-labs.com/se_services.html
Supported Languages	Java
Supported Platforms Where Tool Runs	Java
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Bytecode analysis
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	Unknown
Ability to Discover Debug Code	Unknown
Ability to Discover Unused Code	Unknown
Tool uses CWE Definitions of Vulnerabilities	Unknown
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Unknown
Ability of Testers to Add New Rule Bases	Unknown
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Unknown
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial Service
Licensing	
Vendor Technical Support	Unknown
Vendor Services / Professional services support	Yes
Required training or experience level to operate	Unknown
Vendor provided (or 3rd party provided) training available	Unknown
Comments	

Product	UNO
Description	Identifies uninitialized variables, null-pointers, and out-of-bounds array indexing and "allows for the specification and checking of a broad range of user-defined properties". Aims for a very low false-positive rate..
URL	http://spinroot.com/uno/
Supported Languages	C
Supported Platforms Where Tool Runs	Linux, Unix, Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Free
Licensing	Custom
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Veracode (formerly @stake's analyzer)
Description	Veracode provides a suite of tools for static and dynamic analyses of applications
URL	http://www.veracode.com
Supported Languages	N/A
Supported Platforms Where Tool Runs	Veracode.com
Supported Platform Where Target Resides	Veracode.com
Supported Compilers	N/A
Can Tool be used Remotely?	Yes
Finds or Checks for: (Tool Category)	Binary Code analysis
Lifecycle Position(s)	Testing, Acquisition
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	Yes
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial Service
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	Low
Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Visustin
Description	An automated program for generating flow charts. Visustin can be used to create flowcharts and UML activity diagram-style charts from source code. Visustin accepts source code files as input, and supports a total of 31 programming languages. Supported languages include Ada, ASP, assembler, BASIC, C/C++, C#, Clipper, COBOL, ColdFusion, Fortran, Java, JSP, JavaScript, LotusScript, Pascal/Delphi, Perl, PHP, PL/SQL, PowerScript, PureBasic, Python, QuickBASIC, REALbasic, T-SQL, VB, VBA, VB.NET, Visual FoxPro, and XSLT. Visustin creates a flow chart or an activity diagram using automated layout routines. The user can also draw a flow chart manually. The resulting flow chart can be printed, saved in multiple bitmap and vector formats, or can be exported to external programs (e.g., Microsoft Word, Visio, and PowerPoint). Visustin provides several source code metrics, including McCabe cyclomatic complexity and Decision density
URL	http://www.aivosto.com/visustin.html
Supported Languages	Ada, ASP, BASIC, C/C++, C#, Clipper, COBOL, ColdFusion, Fortran, Java, J#, JSP, JavaScript, LotusScript, MASM assembler, MSP430 assembler, NASM assembler, Pascal/Delphi, Perl, PHP, PL/SQL, PowerScript, PureBasic, Python, QuickBASIC, REALbasic, T-SQL, VB.NET, VBA, Visual Basic, Visual FoxPro, XSLT
Supported Platforms Where Tool Runs	Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Risk Analysis
Lifecycle Position(s)	Design, Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	N/A
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	N/A
Ability of Testers to Modify Existing Rule Bases	N/A
Ability of Testers to Add New Rule Bases	N/A
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	N/A
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	No
Required training or experience level to operate	High

Vendor provided (or 3rd party provided) training available	No
Comments	

Product	Viva64
Description	Finds problems in porting to 64-bit architecture, e.g. out-of-bounds indexing or arithmetic overflow. Integrates with Visual Studio.
URL	www.viva64.com/
Supported Languages	C++
Supported Platforms Where Tool Runs	Windows 64-bit
Supported Platform Where Target Resides	
Supported Compilers	MSVS
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	None
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	No
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	Yes
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	Xception
Description	A binary fault injection tool that simulates an embedded processing environment.
URL	http://www.xception.org/
Supported Languages	N/A
Supported Platforms Where Tool Runs	Linux, Solaris, Windows
Supported Platform Where Target Resides	Linux, LynxOS, Windows
Supported Compilers	N/A
Can Tool be used Remotely?	Yes
Finds or Checks for: (Tool Category)	Fault Injection
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	N/A
Ability to Discover Debug Code	N/A
Ability to Discover Unused Code	N/A
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	No
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	No
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Commercial
Licensing	
Vendor Technical Support	Yes
Vendor Services / Professional services support	Yes
Required training or experience level to operate	High
Vendor provided (or 3rd party provided) training available	Yes
Comments	

Product	Yasca (Yet Another Source Code Analyzer)
Description	A plugin-based framework for scanning arbitrary file types, that aggregates FindBugs, PMD, Jlint, antiC, and Lint4j
URL	http://www.yasca.org/
Supported Languages	C, C++, Java
Supported Platforms Where Tool Runs	Windows
Supported Platform Where Target Resides	
Supported Compilers	N/A
Can Tool be used Remotely?	No
Finds or Checks for: (Tool Category)	Source Code Analyzer
Lifecycle Position(s)	Testing
Scalability (Ability to scan up to 1,000,000 LOC?)	Unknown
Ability to Identify Comments in Code	No
Ability to Discover Debug Code	No
Ability to Discover Unused Code	No
Tool uses CWE Definitions of Vulnerabilities	No
Frequency of Rule Base Updates by Tool Provider	Unknown
Ability of Testers to Modify Existing Rule Bases	Yes
Ability of Testers to Add New Rule Bases	Yes
Ability to provide suggestions for mitigating vulnerabilities (Remediation). If able, is it Active or Passive?	Yes (Passive)
Cost (Hourly/ Flat Fee) [AVAILABILITY]	Open Source
Licensing	Various
Vendor Technical Support	No
Vendor Services / Professional services support	No
Required training or experience level to operate	Medium
Vendor provided (or 3rd party provided) training available	No
Comments	