Automated Detection

This is a stub of a page.

These are comments from Lori Flynn, suggesting more description of capabilities and limits of automated detection:

An understanding of the capabilities and limits of automated detection will help readers of this standard to better use the coding rules and guidelines.

There are three types of problems that static analysis can analyze for:

1. syntactic;
2. semantic; and
3. depend on the intention of the programmer.

Automatic detection of syntactic problems can be 100% correct. Automatic detection methods for semantic problems can do well but cannot guarantee 100% detection because of the Halting Problem. Automatic detection methods for problems that depend on the intention of the programmer cannot be 100% correct, and must depend on heuristics that attempt to intuit the intention of the programmer. For examples of automated detection of the third type of problem, see the paper Automated Code Repair Based on Inferred Specifications (by Klieber and Snavely in the SEI CERT Secure Coding group), that describes automated repairs for three types of bugs: integer overflows, missing array bounds checks, and missing authorization checks. Another example of automated detection of the third type of problem is that one static analysis tool provides an alert if in the analyzed codebase it detects that 4 of 5 times the code checks a particular function's return value to see if it is null, but in one case the return value isn't checked to see if it is null. In this way, it is inferring programmer intent from the other return value checks. A final example of automated detection inferring intent of the programmer is that one tool assumes a variable is sensitive if it is named "password" (and uses this inferred sensitivity for taint flow analyses).

By understanding the limits of automated detection for each coding rule in the standard, managers and developers can better use this standard.

Static analysis and automatic code repair tools are highly useful, but both have their limitations, and should be supplemented with additional secure coding lifecycle methods to increase security of the code. For some types of code flaws, automated static analysis still requires human inspection (a.k.a., auditors of the static analysis diagnostics) to determine if the automatically-generated warning is true or false. For other types of code flaws, automated analysis can correctly determine if the problem exists, and some tools also can automatically 'repair' (edit) the code to correct such problems. Some tools can edit code in a way that can be proven to not create new errors, even if the possible code flaw that was identified is not actually a true flaw (see paper Automated Code Repair Based on Inferred Specifications).

Dynamic analysis (including fuzz testing, for instance using SEI's Basic Fuzzing Framework fuzzing tool) can be automated, and can detect and verify some code flaws. Unit testing and regression testing can also be automated, and provide useful checks to a codebase.

For some code flaws, automated detection methods are too costly (take too much time, too much memory, or too much disk space) to be practical. Makers of automated detection tools (both proprietary code analysis tools and cost-free, open-source code analysis tools) must balance including the ability to check for a particular code flaw with the average user's cost, user's interest in finding that code flaw, and the false-positive rate of that particular code-flaw checker. Checkers that have high false-positive rates tend to displease tool users. For detailed discussion of the issues discussed in this paragraph, see the article A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World.

Widely-used automated code flaw detection tools often find somewhat-overlapping but quite different sets of code flaws, even just looking at automated static analysis tools (e.g., see SEI technical note, Improving the Automated Detection and Analysis of Secure Coding Violations). Some code analysis frameworks use multiple analysis tools to analyze code for a wider variety of code flaws, however the number of code warnings (many of which are false positives) that must be manually inspected increases accordingly (for more information on this topic, see SEI blogpost Prioritizing Alerts from Static Analysis to Find and Fix Code Flaws).

Human code review is manual (not automated, although automation can help document findings and schedule reviews), but can detect some errors that widely-used automated static and dynamic analysis tools do not check for.

Software architecture also impacts a codebase's security, and some analyses of software architecture can be automated.