Nathan Quirk  
Article Review 6  
“Automated Code Review Tools for Security”

In “Automated Code Review Tools for Security,” Gary Mcgraw explains why static analysis tools are an essential tool for identifying security risks in software before they hit production code. He asserts that the two most effective practices for building a secure application are code reviews with static analysis tools and architectural risk analysis. Of these, static analysis can be partially automated and as a result provides the most potential bang for your security buck.

McGraw contends that most security flaws are the result of simple errors in the code. These errors are preventable if the developers can catch them before they reach production. All too often, McGraw states, programmers are eager to get the job done, and the process of picking through code for a stray error is challenging and min-numbing. Add to this the fact that programmers are often not trained to think about security and the languages they use often weren’t designed with security in mind. Simple, automated tools are needed to help identify risks.

Enter static analysis tools. Like manual code reviews, static analyzers examine code for known security errors. Just as compilers eliminate the need to examine every line of code for missing semicolons and curly braces, static analyzers perform mundane tasks quickly. This enables frequent checking and higher detection rates. As the name would suggest, static analysis examines the source code and differs from dynamic analysis in that the code doesn’t have to be running.

Past efforts to build static analysis tools simply searched for troublesome code snippets, a functionality the author likens to the Unix utility grep. Then, lexical analysis capabilities were built in, so that tools could differentiate between an actual problem and, say, a part of a comment that wouldn’t be cause for concern. Then, compiler technology began to be built in. Abstract syntax trees were implemented so that language semantics could be incorporated. Finally, analysis scope was fine-tuned, so that examination could take place in a limited scope within a single method or globally.

For a human, trying to keep pace with an ever-expanding list of known software security flaws is daunting. Add to that the difficulty in scouring your code for all of them and it’s easy to see why a static analysis tool can be a life-saver. Additionally, these tools automatically track control flow and call chains to see where interactions between local scopes might contain issues.

The author stresses the need for these errors to be found before the get put into production, as they often remain unnoticed for long periods of time. While security flaws lie dormant, other code is built upon them and when the time finally comes to root out the problem, the fix is an extremely expensive undertaking.

In conclusion, McGraw feels that these tools need to be used regularly and he is glad to see them becoming more widespread in their use.