
Increasing the Reliability of Security Testing Results

WHITE PAPER

1. Introduction

To increase the reliability of security testing results, developers are finding that security testing should be a combination of analysis techniques—utilizing source code analysis information to direct a second, more practical approach called dynamic analysis. This allows developers to identify vulnerabilities more accurately and confidently than with either technique individually. This combination approach, known as hybrid analysis, produces the accurate and reliable security information that developers need to assess the security of their code.

2. The Guessing Game

Source code analysis products use a technique called variable tracing. Developers use these tools to inject test data into the application during security testing to study the software's potential values and behaviors through the call graphs that represent data flows through the application. By injecting test data this way, the source code analysis product infers what behavior may occur for a certain scenario and variable value—some refer to this technology as an inference engine.

The danger of the source code analysis technique is that it produces inferences, or guesses, as to how the system might behave during run-time and production configuration conditions. Source code analysis can only determine possible security vulnerabilities in the application, which usually results in high false positive rates during security testing.

In the security testing field, trusting the inferred results of source code analysis is analogous to trusting that an application will function according to design when it compiles cleanly. If all code is syntactically and semantically correct, then it will compile. But do you have confidence that it will meet the functional requirements simply because it compiled? Similarly, developers who rely on source code analysis to infer security problems in the application must also perform additional security testing in order to validate the application's real run-time behavior with respect to the potential vulnerabilities.

3. The Fuzz

Also known as automated penetration or fuzz testing, dynamic analysis occurs when a security tool actively attacks the running application based on thousands of known vulnerabilities and attack patterns. A dynamic analysis tool executes thousands of hack attempts on the application in a matter of minutes, just as a hacker would over days or weeks.

The danger of taking only the dynamic analysis approach is that it can be less thorough than source code analysis because it does not have access to, or detailed knowledge of, the application source code. Dynamic analysis tools are used during security testing to crawl an application like a Web spider to discover all of its pages and files and then use this site map to direct automated hack attempts. If the tool is unable to "guess" where some pages or files are located, or is blocked by complex authentication or session management, then it would not be able to effectively attack and assess the security of those hidden resources. The developer can then end up with a false sense of security.

4. Conclusion

Consider the example of a cross-site scripting vulnerability whereby an attacker is able to embed malicious code into an application and trick a user into executing the code on their own machine. During security testing, a source code analysis product might be able to identify the potential of a cross-site scripting vulnerability by finding un-validated inputs or poor session handling—if the particular language and compiler is supported. This information is useful to a developer when pinpointing potential problems. But efforts can be misdirected or wasted when developers spend time fixing a potential vulnerability that in reality is not even exploitable in the application.

A hybrid analysis tool, which will know about the cross-site scripting possibility from an analysis of the source code, will target this potential vulnerability during the dynamic analysis phase of security testing. The tool can determine whether the page is exploitable by attempting to hack it. Furthermore, dynamic analysis can also identify vulnerabilities in a third-party component or database code that source code analysis would not uncover, since it doesn't have access to the third-party component's source code.

Developers are beginning to take the important step toward performing security testing before their applications leave their environments. Analysis tools, such as source code analysis or dynamic analysis, are alone only a partial solution. Developers should look toward hybrid analysis tools to help them secure code more easily and confidently.