EXECUTIVE BRIEF

APPLICATION SECURITY TESTING COVERAGE

Four Reasons the Time for Static Application Security Testing (SAST) Has Passed

INTRODUCTION

Coverage is the most critical aspect of your application security strategy, but the word “coverage” itself needs to be unpacked. There are four dimensions of application security testing coverage that have to be considered, each of which is explained in greater detail in Figure 2.

1. **Portfolio Coverage:** Does your testing approach scale out effectively across your application portfolio?
2. **Security Analysis Coverage:** Is your testing robust enough to assess your applications for all the types of vulnerabilities you care about?
3. **Code Coverage:** Do you effectively test all of the executed code that is part of your applications?
4. **Continuous Coverage:** Does your security testing run continuously with development?

In this brief, we leverage this four-dimensional framework to provide a practical comparison between Contrast Assess and Static Application Security Testing tools and examine their respective abilities to deliver coverage.

CONTRAST ASSESS VS. STATIC APPLICATION SECURITY TESTING (SAST)

Before diving into a side-by-side comparison of SAST and Contrast Assess, let's take a qualitative look at how SAST tools stack up compared to Contrast Assess.

SAST performs static code analysis to identify vulnerabilities by attempting to build a model of the application and pseudo-execute it (via scans) to guess what its runtime behavior might look like. But, the promise of static code analysis - finding flaws in code without having to build and run the program, is also the source of the numerous limitations in the SAST approach and primary reason SAST tools are unable to provide comprehensive coverage. This has always been a problem, but is particularly acute when it comes to modern software development.

First, modern applications are mostly composed of third-party frameworks and libraries, and their architecture is heavily designed around the use of APIs. They are assembled at runtime, and extensively leverage dynamic language features (e.g., reflection, IoC/dependency injection, etc.). SAST tools, however, have only limited context into these elements to inform analysis. Third-party components, custom configuration items or changes that are external to code, and other features that modify the application's behavior during runtime, are all outside of the domain of what SAST covers.

Second, modern development teams, increasingly adopt hyper-agile processes, with workflows spanning across multiple, continuous pipelines that dramatically increase the velocity at which applications evolve. In such environments, developers require short feedback loops, and expect accurate results from testing within minutes, not hours or days. The SAST model, however, heavily relies on security experts to effectively implement long-running scans and then triage results. Experts are scarce resources that, more often than not, become bottlenecks in the delivery pipeline. This model does not scale and cannot effectively provide continuous vulnerability assessment.

Each of these constraints presents its own set of challenges to SAST, but they are interrelated in that they adversely affect a SAST tool's ability to provide coverage.

Contrast Assess leverages instrumentation to gain deep visibility into all the different layers of an application, and uniquely combine multiple vulnerability analysis techniques into a single solution. Instrumentation operates unobtrusively during normal use and testing of an application, enabling the application to self-test for vulnerabilities with full runtime context and high accuracy. As security-instrumented software runs, vulnerabilities are continuously and automatically identified and delivered to Development and Security teams to act on. By eliminating scans, Contrast empowers experts to remove themselves from the critical path of software development, while retaining oversight and spending more time on strategic security initiatives. This model delivers the greatest degree of automation and scale to achieve coverage.

Highlighted below is a comparison between SAST and Contrast Assess across the four dimensions of coverage:

Figure 1. Comparison between SAST and Contrast Assess across the four dimensions of coverage

	SAST	CONTRAST
Portfolio Coverage	**Hard to deploy and run at scale** • Significant effort to tune and run effectively • Heavy reliance on human experts to triage results • Scans complicate pipelines and often avoided/ ignored	**Purpose-built to scale** • Fully automated solution, no experts required; applications self-inventory and self-assess • Runs in a distributed manner across pipelines • Developer self-service drives adoption
Security Analysis Coverage	**Limited** • Static code analysis only; no contextual understanding of the running application • Does not effectively handle dynamic language features • Does not effectively handle frameworks • Prone to False Positives and False Negatives	**Comprehensive** • Combines static, dynamic, runtime data/control flow, configuration and composition analyses into a single solution • Instrumentation enables runtime context and visibility into application; wealth of info sources, including HTTP, code, 3rd party libraries, server environment, backend connections etc.
Code Coverage	**Black-box** • Does not cover third-party libraries; does not cover execution paths that go through library code • Misses dynamically loaded/executed code • Does not effectively work on APIs • Static analyzer is a black-box; no way to measure code coverage or influence the paths to be analyzed	**Transparent** • Analyzes the entire application (assembled and running) across all executed code, including custom and open source • Effectively works with web applications and APIs; not constrained by use of dynamic language features • Monitors and reports on route coverage in real-time to provide visibility into what parts of the app were analyzed
Continuous Coverage	**Does not support** • Point-in-time analysis; leaves gaps in between scans • Whole program analysis is required to find serious flaws, but takes too long and is prone to FPs; Lightweight and Incremental scans provide faster results, but are limited in what they can analyze and find, and provide significantly less security coverage	**Continuous by design** • Runs continuously across SDLC, from a dev workstation with pre-commit local testing, through automated and human driven tests in QA/staging environments to production monitoring) • Shortens feedback loops by delivering instantaneous security feedback for early detection and remediation • Agent install is one time-effort; then, agents automatically deploy and update

CONCLUSION

To maximize application security coverage, security leaders should seek a solution that provides the most assurance across the four dimensions of coverage with the least effort. While SAST is a well-established, traditional approach to security testing, it is plagued with constraints limiting its ability to achieve coverage.

Contrast Assess transforms application security testing by making all existing testing work do double-duty as security testing. That means that automated and human-driven tests, executed by application developers and/or dedicated testing teams, and even live production traffic, all deliver a continuous stream of security feedback. This gives organizations a massive leg up in increasing their application security coverage. The key to this is instrumentation, which makes security testing coverage visible at all times, makes it easier to increase coverage, and provides a clear path to achieving security testing completeness with high assurance.

For a deeper dive into this topic, read our technical brief *Mastering Coverage: The Four Dimensions of Application Security Coverage.*

Figure 2. A framework to evaluating your application security testing coverage

The four dimensions of application security testing tool coverage, explained in greater detail.

1. Portfolio Coverage: Does your testing approach scale out effectively across your application portfolio?

Many organizations only test a very small portion of their application portfolio. As a CISO at one of our prospective customers once volunteered "90% of our application portfolio is one click away from the Wall Street Journal", illustrating this disturbing reality. To scale vulnerability assessment across your portfolio, you should remove security experts from the critical path. The most effective method to accomplish that is to embed in your SDLC an automated solution that enables your applications to self-inventory and self-test in a distributed manner across all pipelines. Such a solution should be rolled-out to development and operations teams in a self-serviced way that stays in their existing workflows and integrates into the tools they already use.

2. Security Analysis Coverage: Is your testing robust enough to assess your applications for all the types of vulnerabilities you care about?

Verifying your software is secure, requires vulnerability assessment that delivers both breadth and depth. To effectively achieve that, you should ensure your testing covers all permutations of relevant vulnerabilities, while leveraging the appropriate analysis technique for each type of vulnerability. Moreover, you should consider whether the tools you use even have the information necessary to accurately diagnose these vulnerabilities. Tools that are prone to false positives waste time and destroy the value in testing. False negatives are even more insidious, leaving teams with a false sense of security, which to more risk.

3. Code Coverage: Do you effectively test all of the executed code that is part of your applications?

Code coverage is all about making sure your testing efforts effectively analyze the entire application, including the custom code, dynamic code, frameworks, libraries, deployment configuration and runtime platform. Since modern applications are dynamic and mostly comprise of third party components, you should perform your vulnerability assessment in the context of an assembled and running application, not a pile of source code. Finally, code coverage must be measured, so you know what code paths were actually analyzed - favor tools that provide this visibility.

4. Continuous Coverage: Does your security testing run continuously with development?

Time is a critically important dimension for security in the continuous world of Agile and DevOps. In today's ever-evolving threat landscape and high-velocity CI/CD pipelines, a single glimpse into application security once every few weeks or even a week is simply not enough visibility. Continuous coverage requires an automated solution that not only enables verification across the SDLC, but also provide accurate, real-time stream of security feedback that development teams can immediately take action on. The key here is end-to-end automation to remove bottlenecks and establish short feedback loops, delivering the fast turnaround teams need at each step in the pipeline.