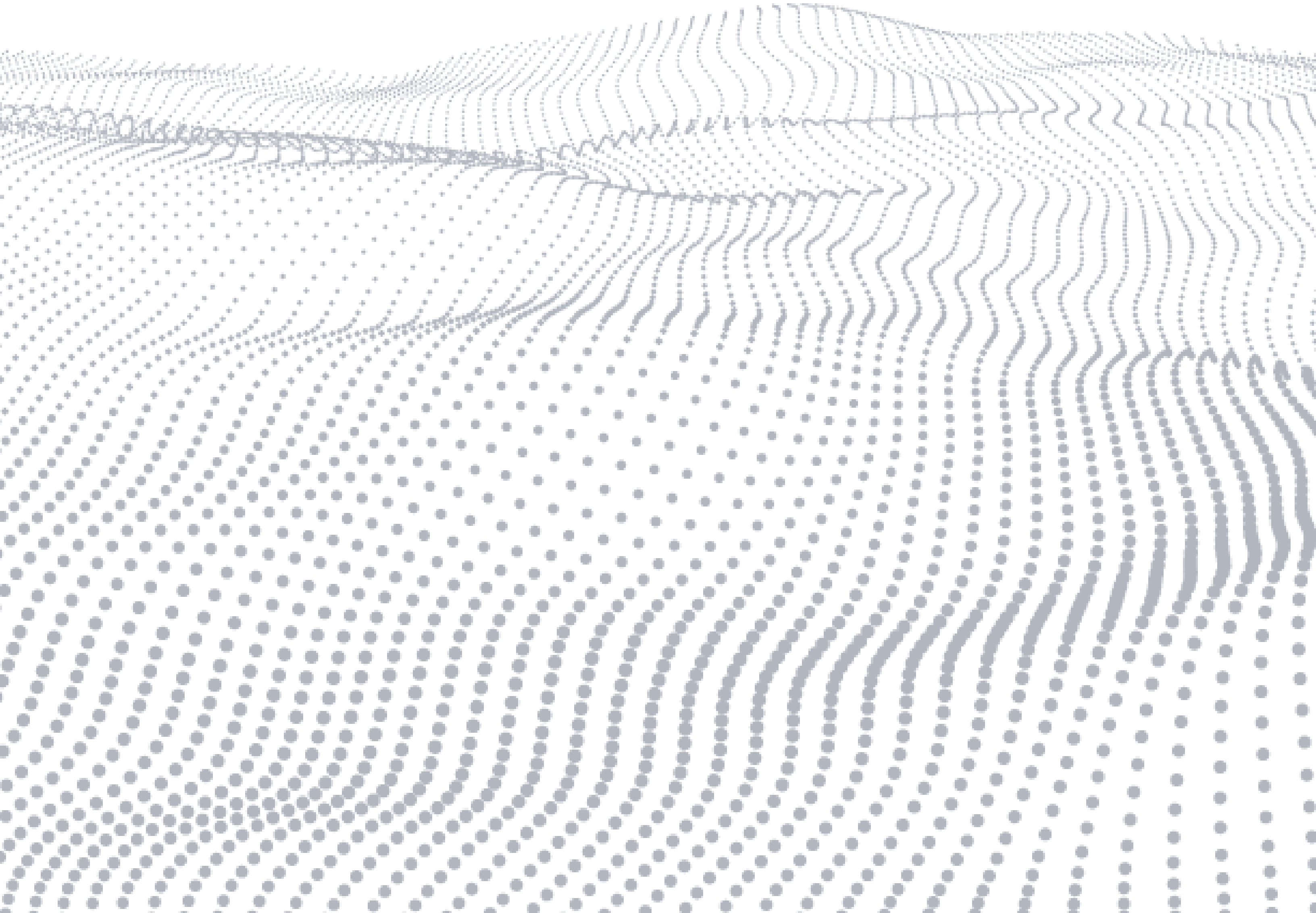
WHITE PAPER

# Breaking the Build in the CI/CD DevOps Life Cycle



## To stop the train, simply pull the chain

As a kid, I often traveled by train in India. I always wondered what would happen if I pulled the chain under the sign that read, “To Stop Train, Pull Chain.” My parents warned me that it would cost them a fortune to pay the fine and that I’d be taken away by the police. Even though it scared me as a child, I was still tempted by the thrill of pulling that chain.

Fast-forward 25 years, and I find myself pushing my clients to pull a similar metaphorical chain. The chain in this case exists in their fast-tracked continuous integration/continuous delivery (CI/CD) pipeline, where the procedural train moves at an increasing speed to support innovation requirements.

Before digging into why pulling the proverbial CI/CD pipeline chain is important, let’s first examine some key software security activities. We’ll explore activities that take place inline within the pipeline, in addition to those performed out-of-band.

“What’s the difference between inline and out-of-band activities?” you may ask. Inline activities are those that you can completely automate and run in a CI/CD pipeline without any human intervention. Examples include static analysis, dynamic analysis, and software composition analysis.

Out-of-band activities are those that you can’t completely automate. Examples include architecture risk analysis, threat modeling, and manual code review.

You can perform most inline activities out-of-band. However, you can’t perform out-of-band activities inline. As an example, there’s currently no way to automate architecture risk analysis or manual code review.

## Building security into the DevOps SDLC

Software security activities can be added throughout the software development life cycle (SDLC). The activities described in this eBook are methodology agnostic. They can be applied to standard SDLC methodologies as well as waterfall, agile, and DevOps methodologies.

It’s critical to include security engineering in the following aspects of the SDLC:

- Software requirements
- Architecture
- Design
- Coding
- Testing
- Validation
- Measurement
- Maintenance

The following out-of-band and inline security activities illustrate how software practitioners can apply security to the various software artifacts produced during software development. You can execute these security activities to improve both the security and quality of the applications you deploy—regardless of your SDLC.

It’s critical to include security engineering across the software development life cycle.

## Out-of-band activities

Threat modeling creates a blueprint that describes your system's attack surface. It does so by identifying major software components, assets, threat agents, security controls, trust zones, and the corresponding relationships between objects.

Architecture risk analysis (ARA) identifies technical risks to your business due to technical flaws in a system's design. The analysis results in specific mitigation and remediation advice for individual defects. The ARA process creates threat models, which are then used to discover architecture-level flaws in software.

Manual secure code review discovers violations of secure coding standards and best practices by reviewing the application's source code line by line. Code reviews are a common mechanism for evaluating the efficacy of security controls and coding constructs that are implemented to satisfy specific security requirements. These reviews are considered manual because humans carry them out.

Penetration testing involves the review of a running application to identify potential security vulnerabilities. Penetration tests generally combine automated tool-assisted testing and in-depth manual analysis focusing on business logic as part of the security assessment of an application or system. As part of this exercise, the tester looks to exploit weaknesses in the application or system to gain access to restricted information and functionality. Penetration testers use the following approaches:

- Black box: Testers have no access to or information about the system. They might use information available in the public domain to build test cases. They might also use various social engineering techniques (e.g., phishing) to gain access to the application or environment.
- Gray box: Testers are provided with a piece of information (e.g., IP addresses, user accounts) for use in the testing process.
- White box: Testers have access to artifacts about the application (e.g., design documents, source code, API specifications) during the testing process.

## Inline activities

Static application security testing (SAST) is a software security activity consisting of automated source code reviews. This static analysis technique uses commercial (e.g., Coverity® Static Analysis) and open source tooling options.

Software composition analysis (SCA) is a process that consists of compiling, tracking, and monitoring free and open source software (FOSS) used by software development teams.

Dynamic application security testing (DAST) is performed during the SDLC testing phase, where the application or system is tested from a black box perspective. DAST is usually a tool-assisted, automated testing process that identifies common security vulnerabilities.

Insider threat detection identifies dangerous code written by malicious in-house developers or outsourced resources. Examples of malicious code:

- Backdoors
- Logic bombs
- Time bombs
- Nefarious communication channels
- Obfuscated program logic
- Dynamic injection code

## Breaking the build: The relationship between security and quality

The purpose of breaking the build is to treat security issues with the same level of importance as quality and business requirements. If quality or security tests fail, the CI server breaks the build and sends notifications. When the build breaks, the CI/CD pipeline also breaks. Depending on the reason for the broken build, it triggers appropriate activities, such as ARA, threat modeling, or manual code review.

If the build broke owing to a critical or high-risk finding (e.g., a SAST tool identified a SQL injection weakness), the development team can resolve the issue, check in the code, and start the next integration. However, if the build broke owing to changes in the API (e.g., the addition of password functionality), this may trigger out-of-band activities.

You may be thinking, "OK, we're ready to break the build, but who's responsible for breaking it and resolving the issues?" Let's find out:

### Security team responsibilities

- Select security activities based on the risk of the application, project, and build data.
- Configure appropriate security tools and integrate them into the build pipeline.
- Define rules on which the build breaks.
- Push results to a common metrics dashboard for reporting. Keep a scoreboard, dashboard, or other dynamic record of outstanding security defects.

### DevOps team responsibilities

- Assist the security team in breaking the build.
- Notify development teams when the build breaks and why.

### Development team responsibilities

- Fix issues when the build breaks.

The three teams coordinate to make sure the process is well defined, tools are properly configured, and developers are ready to fix issues when the build breaks. It may take a few weeks to get the process up and running; in some cases, it can take months to get it right.

## Other CI/CD security integration considerations

| Consideration | Description |
|---|---|
| **Checking scan health** | Check conditions to decide whether a scan is good. Helps with build viability. |
| **Breaking the build** | Continue, pause, or break the build on certain criteria. |
| **Tracking bugs** | Create automated tickets in the defect tracking system. |
| **Handling out-of-band activities** | Determine which out-of-band activities to trigger. |
| **Notifications** | Send continuous notifications via email or Slack. |
| **Shared reusable libraries for CloudBees Jenkins** | Use libraries that are easily configurable, redeployable, and scalable across teams, business units, and the entire organization. |
| **GRC integration** | Deliver results programmatically to a single source for the organization's risk management tool. |
| **Source code management / version control** | Trigger scans on SCM workflows such as pre-commit, pre-receive, and pre-merge/pull requests. |

## What activities can break the build?

All inline activities can break the build, but out-of-band activities can't, because they're carried out by humans, not automated. While you shouldn't break the build for every finding or security issue, here are some points where you should:

- The number of critical vulnerabilities crosses a threshold.
- The number of high-risk vulnerabilities crosses a threshold.
- An application security testing tool identifies a specific vulnerability, such as SQL injection or cross-site scripting (XSS).
- Your SCA tool identifies a zero-day vulnerability (e.g., a Struts 2 critical issue).
- Your SAST scan results indicate that one or more source files failed to compile, thus reducing the confidence in the completeness and validity of the scan results.
- Your SAST or DAST tool runs out of memory.

While this list isn't exhaustive, the processes it outlines help evaluate code strength. They also help you prevent weak or insecure code from remaining undetected until later phases in the CI/CD workflow pipeline.

## Benefits of breaking the build

- Development teams are notified immediately in the case of a failure (e.g., tool failure, high number of findings, or detection of vulnerabilities such as SQL injection or XSS).
- The security team receives alerts on critical and high-risk security issues as soon as they're introduced in the application.
- Breaking the build can trigger out-of-band activities that you don't normally perform.

There is no one-size-fits-all solution to inject security and quality into development workflows. No matter the risks that threaten your applications, it's critical you break the build for security issues just as you do when code doesn't compile or unit tests fail.

So go ahead and pull the chain to stop the build.

The first step is to configure the rule sets to break builds on critical and high-risk issues. One way to ensure that your organization upholds this practice is to incentivize teams to break their builds for security issues in addition to quality issues.

Explore how Black Duck® can help you build security into your DevOps SDLC. Learn more

## About Black Duck

Black Duck® offers the most comprehensive, powerful, and trusted portfolio of application security solutions in the industry. We have an unmatched track record of helping organizations around the world secure their software quickly, integrate security efficiently in their development environments, and safely innovate with new technologies. As the recognized leaders, experts, and innovators in software security, Black Duck has everything you need to build trust in your software. Learn more at www.blackduck.com.