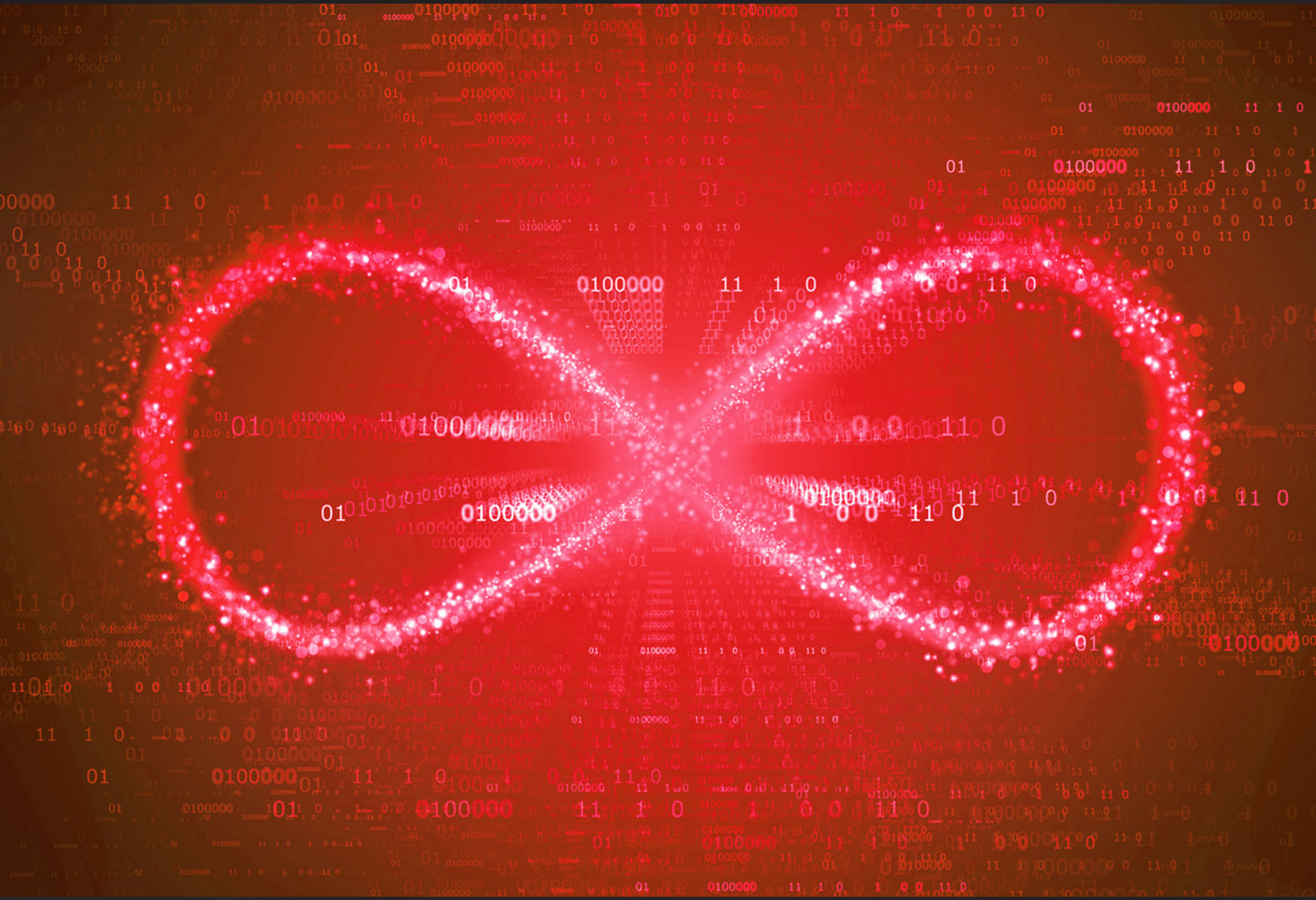
GRAMMATECH

# INTEGRATING STATIC APPLICATION SECURITY TOOLS (SAST) IN DEVSECOPS



TRUSTED LEADERS OF SOFTWARE ASSURANCE AND ADVANCED CYBER-SECURITY SOLUTIONS

WWW.GRAMMATECH.COM

## INTRODUCTION

Software development teams are continually pushed to deliver more complex software systems, including cyber physical systems, in shorter time with less resources. At the same time, the cost of failure is increasing as software is often integrated into larger, more complex business chains such as in the case of IoT edge-to-cloud chains, or interaction between components in a smart-grid deployment or an automotive use case. Teams are continuously trying to improve their tools, methodologies and processes, and this is where DevOps has sprouted from, the combination of software development and systems operations to make sure that software development is not done in a vacuum, but in combination with the teams that operate these systems in the real-world.

The next step in this improvement of software development methods is DevSecOps, where Security is included as a critical part of the development process. The realization here is that a security failure is the same, or worse, as a quality failure. Defects in fielded product impact the bottom line as well as company reputation. It is even worse, if a review after the fact determines that these defects could have easily be avoided. So DevSecOps is the integration at the team level of the teams building the software, operating the software and securing the software.

This paper takes a look at the role of static application security testing tools (SAST) and in particular GrammaTech CodeSonar and how it can be used in DevSecOps and continuous development pipelines to improve quality and security and ultimately, make teams more competitive in getting market leading solutions out the door quicker.

## DEVOPS AND SECURITY

Teams participating in DevOps don't intentionally leave out security but like any other development method, unless it's part of the requirements and development culture, it doesn't get done. Which leads software teams to either ignore security or delegating it to the end of the development effort, which means that people try to 'tack on' security at the end. And as any security practitioner will say, "security needs to be built in from the beginning, not added at the late stages of product development." Or in other words, "pay me now, or much more later."

In addition, a key reason to build security into agile and continuous processes is to build upon the knowledge that accumulates over the project. It's not reasonable to expect software teams to understand their complete attack surface, for example, at the beginning of the project. Building security into day-to-day operations accumulates expertise and knowledge. Starting early is key. With that in mind, DevSecOps is often illustrated as follows on the DevOps flowchart – security in every part of cycle:

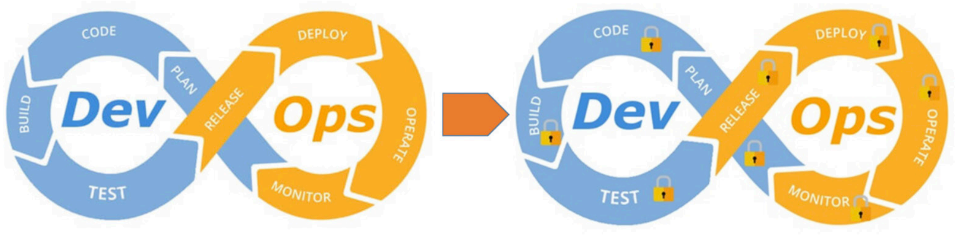


*Figure 1: The DevOps cycle, continuous code, build, test and deploy. DevSecOps is integrating security into all parts of the cycle. Source: "Tripwire - Security at the Speed of DevOps"*

There are two important considerations when adding security to DevOps. The first is security in code, which means, when code is developed, the security of the code itself should be continuously reviewed and assessed. The second is security as code, in other words, security requirements need to be part of the process from the beginning. Let's look at both of these concepts in a bit more detail.

**Security in Code**

Many teams start their path to security off by adopting coding guidelines or standards such as MISRA, or the CERT guidelines, which is a great start, it improves code security from the get-go. However, a coding standard by itself does not avoid memory problems, for example. According to a recent study by Microsoft, 70% of today's security vulnerabilities are caused by memory vulnerabilities. Memory vulnerabilities here could be caused by a simple miscalculation, which led to a buffer-overrun-write, or a more complex path where a variable is tainted through input from the outside that is used without being sanitized. To find these problems in the code more sophisticated analysis is needed such as dataflow analysis and abstract execution.

**Security as Code**

An interesting approach that has come out of DevSecOps practice is the concept of treating security in the same ways as code is; it is guided by requirements, which leads to implementation of security controls, then to validation through testing, and of course requires documentation. It's one of the key ways to integrate security into DevOps and a good way to build security into the development culture and have software teams communicate using a familiar language. Certainly, security implementations that end up as actual code with associated tests can be automated in the same fashion as other code. Automation tools apply here as would any tool that works with requirements, tests, documentation, etc.

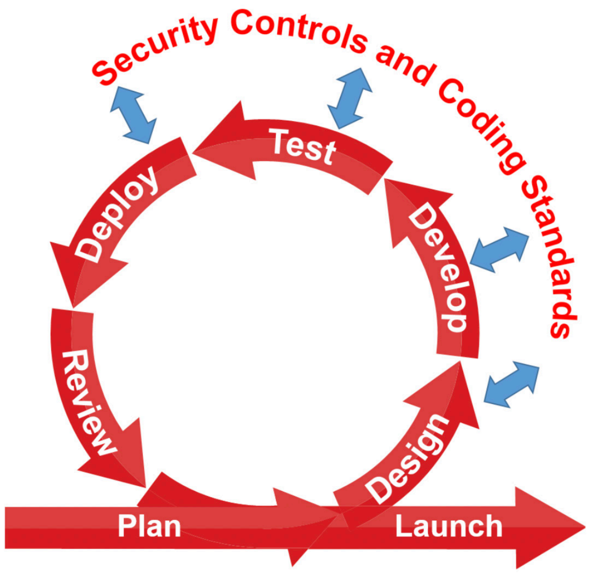


*Figure 2: DevSecOps requires security requirements, controls and coding standards fed into each part of the pipeline. Importantly, feedback is required to close the loop.*

## WORKFLOW INTEGRATION

Automation is one of the main tenants of DevOps. Nothing should be dependent on manual steps as they can be easily overlooked, either by accident or by malicious intent. SAST can be used as soon as code is available in a project and automates the coding compliance and automatic detection of defects and vulnerabilities. In fact, as soon as it's been typed in by a developer – the sooner the better. In the case of legacy, third party and existing code, SAST can be used on those products before even starting the current project, usually run earlier than dynamic analysis, DAST which needs working code, test cases and a test environment.

SAST come in all types of shapes and sizes, some focus on coding standards, some, more advanced tools, step beyond the coding standards and explore data flow, control flow and abstract execution techniques to find defects in the source code that arise from programming or logic mistakes. These advanced tools help achieve security in code and assist in creating security as code.

Continuous integration and deployment processes rely on automation in order to realize the benefits thereof. Without efficient progress through the cycle, the continuous nature of the processes amplifies inefficiencies. All new code has bugs, the challenge teams face is to remove these bugs as early as possible with as little effort as possible. SAST improve code security and quality early in the process and help the developer do this as easily as possible, ideally, integrated into their development environment.

## INTEGRATING SAST IN DEVSECOPS

Most DevSecOps teams have set up some form of workflow where a developer can develop code in their favorite development environment (be that VI, Emacs, Microsoft VS Code or Visual Studio, or any of a number of Eclipse based environments). Coding here means authoring code, compiling it, unit testing and debugging it. As part of this environment, there is usually a capability for the developer to kick off a build. Some projects do local compilations on the developers desktop, others have more formalized and 'locked down' build servers. Once a work item has been completed, it is submitted, and merged back into the mainline codebase. Automatic testing, both dynamic and static, is integrated into this workflow, typically there is some form of sanity, or smoke-test before changes are allowed back into the mainline code base.

A lot of tool integration is needed to make all of these workflows work smoothly. From requirements management and defect tracking (tools like JIRA), to build automation (tools like Jenkins), automated testing and much more.

SAST integrate well with just about any software automation tool chain and development methodology and process. This is mainly due to the fact they can be used locally by developers at their desktop for instantaneous feedback and used to analyze a complete build whether that's done hourly, or whenever. In addition, these tools are completely autonomous since they require no interaction with testers or developers. They are applicable whenever it makes sense to check for bugs and security vulnerabilities in the code:

- **Develop**: This is the critical time to detect any new security vulnerabilities, as soon as developers write new code (or test cases) even before it's submitted to a build or software control system. CodeSonar, for example, presents these vulnerabilities immediately in the developer's IDE just like a compiler warning, providing easy and actionable corrective action (such as vulnerability assessment, root causes and control and data flow traces). Turnaround time is important here, feedback needs to be fast, such that the developer is not bogged down waiting for builds to finish. The integration into the IDE is important here as it lessens the cognitive load on the software developer. He does not need to leave his favorite development environment and can review and assess the static analysis output right there in his IDE. SAST are also essential in this phase for ensuring secure coding standards are met. Violations of the coding standard and potentially dangerous coding practices are flagged for developers to investigate.

- **Test:** This is where all the changes from all developers are brought together for more comprehensive testing, SAST play an important part in that process. Turnaround time in this phase is less critical, hence SAST provides more thorough analysis requiring more compute time, for example, to find concurrency issues or tainted data flows, or spend more time performing pointer analysis.

- **Deploy**: SAST can be used to analyze deployable binaries and libraries. CodeSonar, for example, support binary code analysis for C/C++ code that detects the same types of vulnerabilities as source analysis. This is a good practice to detect bugs introduced during building and deployment of deliverables.

- **Review**: SAST provide reporting and analysis that software teams can use to evaluate individual warnings but also higher-level assessments of application quality and security. Reports from SAST should be part of the cycle assessment and planning for each cycle.

These integrations into the DevSecOps cycle are illustrated below:

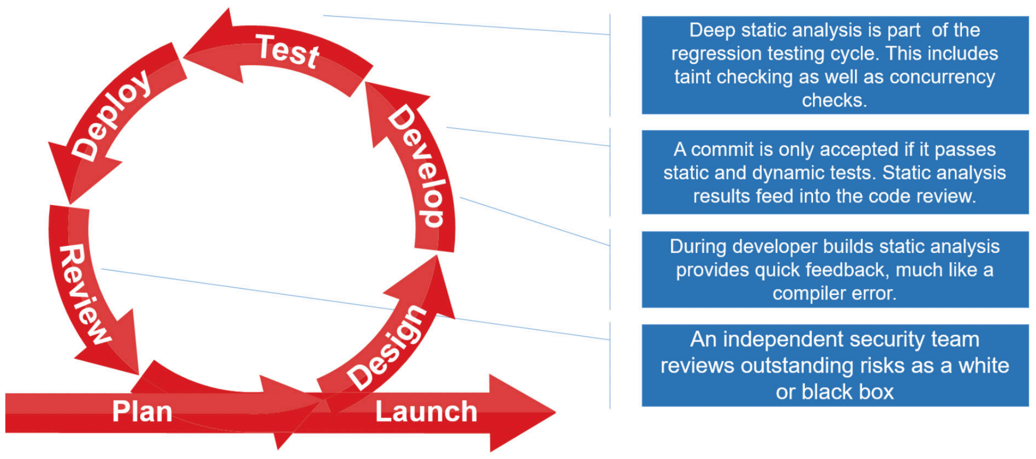

Test
Deploy
Develop
Review
Design
Plan
Launch
Deep static analysis is part of the regression testing cycle. This includes taint checking as well as concurrency checks.
A commit is only accepted if it passes static and dynamic tests. Static analysis results feed into the code review.
During developer builds static analysis provides quick feedback, much like a compiler error.
An independent security team reviews outstanding risks as a white or black box

*Figure 3: The role of SAST in DevSecOps*

## THE ADDED BENEFIT OF BINARY ANALYSIS

GrammaTech CodeConar has the unique ability to perform advanced static analysis on binary code. This provides added benefits to the continuous integration process, especially when incorporating third party binaries or legacy libraries. If source code is not readily available, this does not preclude the ability to detect bugs and security vulnerabilities. In addition, binary analysis can be used by security teams to perform "black box" analysis of product deliverables.

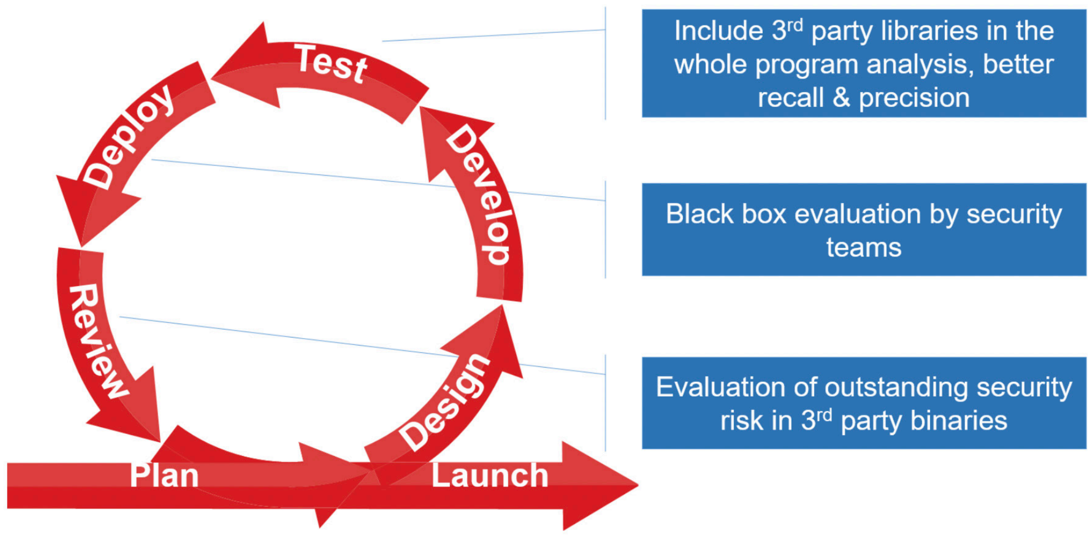

Test
Develop
Deploy
Review
Design
Plan
Launch
Include 3rd party libraries in the whole program analysis, better recall & precision
Black box evaluation by security teams
Evaluation of outstanding security risk in 3rd party binaries

*Figure 4: The added benefit of binary analysis in a continuous integration process.*

### Depth of Analysis

Not all software development teams have the same preference for SAST. The fact is, all of these types of tools suffer from a certain amount of imprecision. Not all warnings that a tool reports are actionable, some are false positives, meaning the tool incorrectly issues a warning. The opposite of false positive is a false negative, an actual problem in the source code that the tool overlooks. Some developers, especially in the enterprise side of software development prefer static analysis tools with a low false positive rate. They do not want to see warnings that are not serious issues. Interestingly enough, there is an inverse relation between false positives and false negatives. Low false positives, typically come with higher false negatives. The cost of a false negative is that a problem could make it into the code base, maybe it is found during testing, but often it makes it to deployed systems.

For safety and security critical software projects missing important defects or security vulnerabilities is clearly unacceptable. Developers working in these projects, would rather spend more time analyzing warnings even if some may turn out to be a false positive, or a less critical problem. Missing a problem and shipping software with a defect could lead to power outages, crashed airplanes or cars, or critical health equipment that malfunctions. For safety and security critical software, a low false negative rate is much more important than a low false positive rate.

CodeSonar is a SAST that is specifically focussed on these safety and security critical projects.

## SUMMARY

Clearly, there is an important role for SAST in the DevSecOps and are an important part of adapting security practices in a continuous integration and deployment pipeline. As software teams start to integrate security into their DevOps, tools such as CodeSonar are easy to adopt and become part of the automation pipeline. By detecting vulnerabilities early and preventing them from entering in later stages of the pipeline pays off in reduced late-stage security fixes.

SAST is an important part of the software development automation tool chain that provides added security vulnerability detection, coding standard enforcement and complementary security assurance to testing and dynamic analysis. CodeSonar in particular, brings the added benefit of binary analysis for black box security assessment of third party libraries and executables.