Сравнительное исследование инструментов статического анализа кода в конвейерах CI/CD

A Comparative Study of Static Code Analysis tools in CI/CD Pipelines

УДК 004
Получено: 19.03.2024 Одобрено: 10.04.2024 Опубликовано: 25.06.2024

Терро Моайад
аспирант, Университет ИТМО
факультет безопасности информационных технологий (ФБИТ)
e-mail: moaed.terro@gmail.com

Terro Moayad
Postgraduate student, ITMO University
Faculty of Information Technology Security (FBIT)
e-mail: moaed.terro@gmail.com

Аннотация
В сфере современной разработки программного обеспечения конвейеры непрерывной интеграции и непрерывного развертывания (CI/CD) служат неотъемлемыми механизмами для поддержания качества, безопасности и эффективности кода. Инструменты статического анализа кода играют ключевую роль в этих конвейерах, автоматизируя обнаружение потенциальных уязвимостей и обеспечивая соблюдение стандартов кодирования. В этом сравнительном исследовании оцениваются и сравниваются ведущие инструменты статического анализа кода, используемые в конвейерах CI/CD, с упором на их возможности с точки зрения политик по умолчанию и настраиваемых политик, интеграции со средами разработки, форматов вывода и возможностей настройки. Анализируя и сравнивая такие инструменты, как KICS, tfsec, Trivy, Terrascan, Checkov и Semgrep OSS, это исследование направлено на то, чтобы дать представление об их сильных и слабых сторонах, помогая практикам принимать обоснованные решения для повышения качества и безопасности программного обеспечения на протяжении всего жизненного цикла разработки. Это исследование подчеркивает важность выбора подходящих инструментов на основе требований конкретного проекта, подчеркивая, что упреждающие меры безопасности в рабочих процессах CI/CD значительно повышают безопасность инфраструктуры и соответствие требованиям.
Ключевые слова: DevSecOps, конвейеры CI/CD, инфраструктура как код, статический анализ кода, сканирование безопасности, инструменты с открытым исходным кодом, обнаружение уязвимостей, SAST, безопасность конвейеров.

Abstract
In the realm of modern software development, Continuous Integration and Continuous Deployment (CI/CD) pipelines serve as integral mechanisms for maintaining code quality, security, and efficiency. Static code analysis tools play a pivotal role within these pipelines by automating the detection of potential vulnerabilities and enforcing coding standards. This comparative study evaluates and contrasts leading static code analysis tools utilized in CI/CD pipelines, focusing on their capabilities in terms of default and custom policies, integration with development environments, output formats, and customization options. By analyzing and comparing tools such as KICS, tfsec, Trivy, Terrascan, Checkov, and Semgrep OSS, this study

aims to provide insights into their strengths and limitations, aiding practitioners in making informed decisions to enhance software quality and security throughout the development lifecycle. This research underscores the importance of selecting appropriate tools based on project-specific requirements, emphasizing that proactive security measures within CI/CD workflows significantly bolster infrastructure safety and compliance.
Keywords: DevSecOps, CI/CD Pipelines, Infrastructure as Code, Static Code Analysis, Security Scanning, Open-Source Tools, Vulnerability Detection, SAST, Pipelines security.

Introduction

In the era of rapid software development and deployment, maintaining robust code quality and security standards is paramount. Continuous Integration and Continuous Deployment (CI/CD) pipelines have revolutionized the software development lifecycle by enabling frequent and automated software builds, tests, and deployments. Within these pipelines, static code analysis (SCA) tools play a crucial role in ensuring that code adheres to best practices, security guidelines, and regulatory requirements before it is integrated into production environments.

Static Code Analysis (SCA) tools analyze code without executing it, focusing on identifying potential vulnerabilities, coding errors, and deviations from coding standards. They provide developers and DevOps teams with early feedback on issues such as security vulnerabilities, performance bottlenecks, and compliance violations. By integrating SCA into CI/CD pipelines, organizations can detect and resolve issues early in the development process, thereby reducing the cost and effort required for fixing defects in later stages. The landscape of SCA tools is diverse, with each tool offering unique features and capabilities tailored to different programming languages, frameworks, and deployment environments. This diversity necessitates a comprehensive evaluation to determine which tools best meet the specific needs of an organization's development and operational workflows[1].

This comparative study focuses on evaluating prominent SCA tools used in CI/CD pipelines, specifically examining their capabilities, integration possibilities, output formats, and customization options. The tools under scrutiny include KICS, tfsec, Trivy, Terrascan, Checkov, and Semgrep OSS, chosen for their popularity, community support, and relevance in modern software development practices. The objectives of this study are twofold: firstly, to provide a detailed comparison of these tools based on their functionalities and features; and secondly, to offer insights into their applicability and effectiveness in enhancing software security, compliance, and development efficiency within CI/CD environments. by conducting this comparative analysis, this study aims to assist software development teams, DevOps engineers, and security professionals in making informed decisions regarding the selection and implementation of SCA tools that align with their organizational goals and technical requirements. Ultimately, the goal is to contribute to the advancement of secure and efficient software development practices in the context of CI/CD pipelines.

Code analysis tools

The world of software development is a diverse one, with each programming language boasting its own strengths and, unfortunately, its own set of vulnerabilities. To combat these language-specific threats, developers have access to a rich arsenal of static code analysis tools. This study delves into the capabilities of various open-source static code analysis tools, offering insights into their effectiveness in detecting vulnerabilities within specific programming languages.

KICS (Keeping Infrastructure as Code Secure)

KICS is an open-source community project and contributions are welcome from security experts and developers to help improve the tool[2,3]. It provides a flexible way to integrate IaC security scanning into DevOps workflows without slowing down software delivery[3]. It

focuses on detecting security vulnerabilities, compliance issues, and best practice violations within IaC scripts before deployment. It offers a comprehensive set of 663 default policies written in OPA Rego language, covering a wide range of security checks and compliance standards across various cloud platforms and infrastructure providers. Users can create custom policies using OPA Rego syntax to tailor security checks based on specific project requirements or internal policies. KICS seamlessly integrates into CI/CD pipelines, supporting Docker, IDEs (such as VSCode), CI/CD systems (including GitHub Actions, GitLab, Terraform Cloud), and Git Hooks, facilitating automated security checks throughout the development lifecycle. It supports a variety of output formats including ASFF, CSV, JSON, HTML, and SARIF, enabling flexible reporting and integration with existing DevSecOps tools. KICS also allows for targeted scans, ignore policies, severity thresholds, and configuration files to customize scanning behavior and enhance adaptability to different project environments. KICS currently lacks robust support for module scanning beyond some public modules from the Terraform registry. This limitation restricts its capability to effectively scan local or private custom modules.

Tfsec

tfsec is another open-source static code analysis tool tailored for Terraform configurations. It focuses on identifying security issues and best practice violations in Terraform scripts to ensure secure infrastructure deployment[4]. tfsec provides a set of 154 default policies written in OPA Rego, covering common security pitfalls and misconfigurations in Terraform code. Like KICS, tfsec supports custom policies using OPA Rego syntax, allowing users to define additional security checks or modify existing ones. tfsec integrates well with Docker, popular IDEs (such as VSCode, JetBrains, Vim), and CI/CD systems (including GitHub Actions). However, it does not support Git Hooks for direct integration into version control workflows. It supports outputs in formats like Checkstyle, CSV, JSON, HTML, Markdown, and SARIF, facilitating comprehensive reporting and integration with various CI/CD pipelines. tfsec offers features for targeted scans, ignore policies, severity thresholds, configuration files, and Terraform variables interpolation, enabling fine-tuning of scanning parameters for specific project needs. While tfsec covers a broad range of Terraform providers and offers extensive customization options, its maintenance focus has shifted towards Trivy, potentially affecting long-term support and updates.

Trivy

Trivy is primarily known as a container vulnerability scanner but also supports scanning for Terraform templates. It identifies vulnerabilities and misconfigurations in container images and infrastructure configurations. Trivy includes 322 default policies written in OPA Rego, focusing on security checks for Terraform configurations along with container images. Like KICS and tfsec, Trivy allows users to define custom policies to extend or modify the default security checks according to specific requirements. Trivy supports integration with Docker, popular IDEs (such as VSCode, JetBrains, Vim), and CI/CD systems (including Azure DevOps, GitHub Actions, Buildkite). It lacks direct support for Git Hooks. It provides outputs in formats such as ASFF, Cosign, CycloneDX, JSON, SARIF, and SPDX, ensuring compatibility with various DevSecOps workflows and tools. Trivy supports targeted scans, ignore policies, severity thresholds, configuration files, and Terraform variables interpolation, enhancing flexibility and adaptability in security scanning setups. Like tfsec, Trivy lacks native support for Git Hooks, which may require additional setup for seamless integration into version control systems.

Terrascan

Terrascan, now maintained by Tenable, is an open-source static code analysis tool specifically designed for scanning Terraform configurations. It focuses on identifying security risks and compliance issues in Infrastructure as Code. Terrascan boasts a robust set of 790 default policies written in OPA Rego, covering a wide spectrum of security checks and compliance standards across multiple cloud platforms and infrastructure providers. Users can

create custom policies using OPA Rego syntax to define additional security checks or modify existing ones to align with specific project requirements. Terrascan integrates with Docker, popular IDEs (such as VSCode), and CI/CD systems (including GitHub Actions, Atlantis). It also supports Git Hooks for direct integration into version control workflows. It supports outputs in JSON, JUnit, SARIF, XML, and YAML formats, ensuring compatibility with various CI/CD pipelines and reporting tools. Terrascan provides features for targeted scans, ignore policies, severity thresholds, configuration files, and Terraform variables interpolation, offering extensive customization options to optimize security scanning workflows. Tarascan's focus on Terraform scanning might limit its applicability for environments heavily reliant on other Infrastructure as Code frameworks or configurations.

Checkov

Checkov, acquired by Prisma Cloud (Palo Alto Networks), is a popular open-source static code analysis tool that supports scanning for infrastructure as code configurations across various cloud providers. Checkov boasts an extensive library of 2110 default policies, covering comprehensive security checks, compliance standards, and best practices for Terraform, Kubernetes, and other IaC frameworks. Checkov supports custom policy creation using YAML and Python, offering flexibility to define and extend security checks based on specific organizational requirements. Checkov integrates with Docker, popular IDEs (such as VSCode, JetBrains), and CI/CD systems (including GitHub Actions, GitLab). It also supports Git Hooks for seamless integration into version control workflows. It provides outputs in formats like CSV, CycloneDX, JSON, JUnit, SARIF, and SPDX, facilitating detailed reporting and integration with various DevSecOps tools and platforms. Checkov offers features for targeted scans, ignore policies, severity thresholds, configuration files, and Terraform variables interpolation, enabling precise customization of scanning parameters for different project environments. While Checkov excels in its broad coverage of default policies and customization options, the complexity of Python-based custom policies may require additional expertise for implementation and maintenance.

Semgrep

Semgrep OSS Open-Source Edition is a static analysis tool that supports scanning for a variety of programming languages and configurations, including Infrastructure as Code scripts. Semgrep OSS includes 362 default rules that cover security vulnerabilities, coding best practices, and potential performance issues across multiple languages and frameworks, including Terraform. Policies in Semgrep are defined using YAML syntax, offering a straightforward approach for creating and modifying security rules to meet specific project requirements. Semgrep integrates with Docker, popular IDEs (such as VSCode, JetBrains, Emacs, Vim), and CI/CD systems (including GitLab). However, it does not natively support Git Hooks. It supports outputs in formats like Emacs, GitLab SAST, JSON, JUnit, SARIF, and Vim, providing flexibility for integration with various CI/CD pipelines and reporting tools. Semgrep offers features for targeted scans and severity thresholds, but lacks support for configuration files and module scanning, limiting some customization options compared to other tools. Semgrep strength lies in its flexibility across multiple programming languages, but its limited support for Terraform module scanning, and configuration files may impact its suitability for complex IaC environments. Table 1 Summarize the differences between reviewed static tool analysis.

Table 1 - Summary of Static code analysis tools comparison

Tool	Integration	Output Formats	Customization Features	Novelty/Contribution
KICS	Docker, IDE, CI/CD, Git Hook	ASFF, CSV, Code Climate, CycloneDX, GitLab SAST, HTML, JSON, JUnit, PDF, SARIF, SonarQube	Targeted Scans, Ignore Policies, Min Severity, Config File, Variables interpolation, Module Scanning	Early adoption for Infrastructure as Code (IaC) security scanning
tfsec	Docker, IDE, CI/CD	Checkstyle, CSV, HTML, JSON, JUnit, Markdown, SARIF	Targeted Scans, Ignore Policies, Min Severity, Config File, Variables Interpolation, Module Scanning	Strong focus on simplicity and ease of integration
Trivy	Docker, IDE, CI/CD	ASFF, Cosign, CycloneDX, JSON, SARIF, SPDX	Targeted Scans, Ignore Policies, Min Severity, Config File, Variables Interpolation, Module Scanning	Specialized in container image security scanning
Terrascan	Docker, IDE, CI/CD, Git Hook	JSON, JUnit, SARIF, XML, YAML	Targeted Scans, Ignore Policies, Min Severity, Config File, Variables Interpolation, Module Scanning	Comprehensive support for multi-cloud environments
Checkov	Docker, IDE, CI/CD, Git Hook	CSV, CycloneDX, GitLab SAST, JSON, JUnit, SARIF, SPDX	Targeted Scans, Ignore Policies, Min Severity, Config File, Variables Interpolation, Module Scanning	Extensive policy coverage and flexible policy creation
Semgrep OSS	Docker, IDE, CI/CD, Git Hook	Emacs, GitLab SAST, JSON, JUnit, SARIF, Vim	Targeted Scans, Ignore Policies, Min Severity, Config File, Variables Interpolation, Module Scanning	Innovative use of semantic grep for code analysis

Related Works

The paper by Emanuelsson P, Nilsson U [5] provides a survey and comparison of three leading industrial static code analysis tools: PolySpace Verifier, Coverity Prevent, and Klocwork K7. The authors aim to identify significant static analysis capabilities beyond what a normal compiler provides and examine the underlying supporting technologies. Some of the comparisons and assessments made by the authors are subjective. For example, determining which tool has better scalability or precision is open to interpretation without clear benchmarks.

Zampetti et al. [6] examined the usage of static code analysis tools within the continuous integration (CI) pipelines of 20 open-source Java projects hosted on GitHub. It found that the most common issues detected were related to coding standards and missing licenses, rather than potential bugs or vulnerabilities. Some projects used a "softer" mode for static analysis that raised warnings but did not fail the build. When builds did break due to static analysis, developers quickly fixed the underlying issues. The study confirmed that developers are often reluctant to configure static analysis tools, highlighting the need for better default configurations.

Anum Fatima et al. [7] evaluated the capabilities of several leading static code analysis tools for C/C++, including Cppcheck, Clang Static Analyzer, Infer, and PVS-Studio. The tools were assessed on their ability to detect common programming issues like injection problems, input issues, and variable/pointer problems. The study found that the tools had varying levels of effectiveness, with no single tool able to comprehensively detect all the problems tested. The choice of static analysis tool should be based on the specific needs and requirements of the project. The results may not generalize to real-world, large-scale codebases, and ongoing evaluation is needed as the tools continue to evolve.

Novak J et al. [8] presents a taxonomy for classifying and comparing static code analysis tools based on their analysis techniques, scope, and objectives. The taxonomy consists of three main dimensions: analysis techniques (e.g. data flow, pattern matching), analysis scope (e.g. whole program, individual functions), and analysis objectives (e.g. security vulnerabilities, coding standards). Taxonomy provides a structured way to evaluate and select static analysis tools based on specific project requirements. It can also guide future tool development efforts. The taxonomy is based on existing literature and tools, so it may not fully capture emerging trends in static analysis technology.

Several studies have evaluated popular static analysis tools like Cppcheck, Clang, Infer, FindBugs, and PMD for their ability to detect security vulnerabilities in C/C++ and Java code. Kaur A, Nayyar R [9] in their studies found the tools had varying effectiveness, with no single tool able to comprehensively detect all vulnerabilities. Tool choice should be based on project needs, as the tools have different strengths and weaknesses. Using a combination of tools can improve coverage. Limitations include the studies being based on limited test cases, so results may not fully generalize to real-world codebases. Ongoing evaluation is needed as the tools evolve.

Conclusion

This comparative study on static code analysis tools has provided invaluable insights into safeguarding infrastructure as code within CI/CD pipelines. The key is that there is no one-size-fits-all solution - the optimal tool depends on unique project requirements, with the ability to balance default security checks and customization emerging as a hallmark of the most versatile options. Seamless integration and flexible outputs further enhance a tool's utility. Proactive security measures within CI/CD workflows can dramatically reduce risks, streamline development, and mitigate costly incidents. As the infrastructure as code landscape evolves, ongoing research will guide practitioners in selecting the right tools and drive the development of even more robust solutions.

REFERENCES

1. Jasper Kathrine G. COMPARATIVE ANALYSIS OF SUBDOMAIN ENUMERATION TOOLS AND STATIC CODE ANALYSIS // JOURNAL OF MECHANICS OF CONTINUA AND MATHEMATICAL SCIENCES. 2020. Vol. 15, № 6.
2. KICS - Keeping Infrastructure as Code Secure [Electronic resource].
3. KICS - Open-Source Infrastructure as Code | Checkmark [Electronic resource]. URL: https://checkmarx.com/product/opensource/kics-open-source-infrastructure-as-code-project/ (accessed: 26.06.2024).
4. Aqua Security. A static analysis security scanner for your Terraform code [Electronic resource] // https://aquasecurity.github.io/tfsec/v1.28.1/.
5. Emanuelsson P., Nilsson U. A Comparative Study of Industrial Static Analysis Tools // Electron Notes Theor Comput Sci. 2008. Vol. 217. P. 5–21.
6. Zampetti F. et al. How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines // 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE, 2017. P. 334–344.
7. Fatima A., Bibi S., Hanif R. Comparative study on static code analysis tools for C/C++ // 2018 15th International Bhurban Conference on Applied Sciences and Technology (IBCAST). IEEE, 2018. P. 465–469.
8. Novak J., Krajnc A., Žontar R. Taxonomy of static code analysis tools // The 33rd International Convention MIPRO. 2010. P. 418–422.
9. Kaur A., Nayyar R. A Comparative Study of Static Code Analysis tools for Vulnerability Detection in C/C++ and JAVA Source Code // Procedia Comput Sci. 2020. Vol. 171. P. 2023–2029.

Список использованной литературы

1. Джасфер Кэтрин Г. СРАВНИТЕЛЬНЫЙ АНАЛИЗ СРЕДСТВ ПЕРЕЧИСЛЕНИЯ ПОДОБЛАСТЕЙ И СТАТИЧЕСКОГО АНАЛИЗА КОДА // ЖУРНАЛ МЕХАНИКИ СРЕЗНЫХ СРЕД И МАТЕМАТИЧЕСКИХ НАУК. 2020. Том. 15, № 6.
2. KICS – Обеспечение безопасности инфраструктуры как кода [Электронный ресурс].
3. KICS — Инфраструктура с открытым исходным кодом как код | Checkmarx [Электронный ресурс]. URL: https://checkmarx.com/product/opensource/kics-open-source-infrastructure-as-code-project/ (дата обращения: 26.06.2024).
4. Аква Безопасность. Сканер безопасности статического анализа вашего кода Terraform [Электронный ресурс] // https://aquasecurity.github.io/tfsec/v1.28.1/.
5. Эмануэльссон П., Нильссон У. Сравнительное исследование инструментов промышленного статического анализа // Электронные заметки Theor Comput Sci. 2008. Том. 217. С. 5–21.
6. Зампетти Ф. и др. Как проекты с открытым исходным кодом используют инструменты статического анализа кода в конвейерах непрерывной интеграции // 14-я Международная конференция IEEE/ACM по репозиториям программного обеспечения для майнинга (MSR), 2017 г. IEEE, 2017. С. 334–344.
7. Фатима А., Биби С., Ханиф Р. Сравнительное исследование инструментов статического анализа кода для C/C++ // 15-я Международная Бхурбанская конференция по прикладным наукам и технологиям (IBCAST), 2018 г. IEEE, 2018. С. 465–469.

8. Новак Ю., Крайнц А., Жонтар Р. Таксономия инструментов статического анализа кода // 33-я Международная конвенция MIPRO. 2010. С. 418–422.
9. Каур А., Найяр Р. Сравнительное исследование инструментов статического анализа кода для обнаружения уязвимостей в исходном коде C/C++ и JAVA // Procedia Comput Sci. 2020. Том. 171. С. 2023–2029.