НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ МОЛОДИХ ВЧЕНИХ

Актуальні проблеми інформаційних технологій



АРІТ



23 жовтня 2026 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ МОЛОДИХ ВЧЕНИХ
«Актуальні проблеми інформаційних технологій»

23 жовтня 2026 року

Матеріали доповідей

Київ 2026

Герасименко О.Ю., Вахнюк М.С., Люлько С.С. AGENTIC DEVOPS ТА МЕЖІ АВТОНОМНОСТІ AI-АГЕНТІВ У КЕРУВАННІ ІНФРАСТРУКТУРОЮ 37
Лещенко О.О., Бікчентаєв І.О. ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ AI-АСИСТЕНТІВ У САМОСТІЙНОМУ НАВЧАННІ СТУДЕНТІВ 39
Махович О.І., Гомельський С.А. РОЗРОБКА ЛОКАЛЬНОЇ RAG-СИСТЕМИ ДЛЯ СЕМАНТИЧНОГО ПОШУКУ ТА АНАЛІЗУ ВИХІДНОГО КОДУ 41
СИСТЕМИ ТА МЕТОДИ ЗАХИСТУ КОМП'ЮТЕРНОЇ ІНФОРМАЦІЇ 44
Герасименко О.Ю., Кичата С.В., Чернега К.В. МЕТОДИ КОНТЕКСТНОЇ ПРІОРИТИЗАЦІЇ ВРАЗЛИВОСТЕЙ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: CVSS, EPSS, SSVC ТА ГРАФОВІ МОДЕЛІ РИЗИКУ 45
Maksymenko D., Makhovych I. METHODOLOGY FOR AUTOMATED REST API SECURITY ANALYSIS WITHIN CONTINUOUS INTEGRATION AND CONTINUOUS DEPLOYMENT (CI/CD) PIPELINES 48
ТЕОРЕТИЧНІ АСПЕКТИ КОМП'ЮТЕРНИХ НАУК 51
Поліщук В.Л., Скідан В.В. КОНТРОЛЬ ВХІДНИХ УМОВ У ПОРІВНЯЛЬНОМУ ОЦІНЮВАННІ МУЛЬТИМОДАЛЬНИХ МОДЕЛЕЙ ЧЕРЕЗ ХМАРНІ API 52
ІНТЕРНЕТ РЕЧЕЙ 54
Кульчицький Д. О., Кравченко Ю. В. ІНТЕЛЕКТУАЛЬНА ОПТИМІЗАЦІЯ МАРШРУТИЗАЦІЇ В КОМП'ЮТЕРНИХ МЕРЕЖАХ KDN-АРХІТЕКТУРИ 55
Ольховець Н.С., Дахно Н.Б. ВІД СЕНСОРА ДО МІКРОКОНТРОЛЕРА: ОГЛЯД ЛОКАЛЬНИХ ПРОТОКОЛІВ ОБМІНУ ДАНИМИ (I2C, UART, 1-WIRE) У СИСТЕМАХ ЕКОЛОГІЧНОГО МОНІТОРИНГУ РХБЗ 57
Гладка М.В., Гладка М.В. СИСТЕМА РОЗУМНОГО ДОМУ НА БАЗІ ESP32 ТА ПРОТОКОЛУ MQTT 59
ІМЕННИЙ ПОКАЖЧИК 61

UDC 004.41
[1] ***D. Maksymenko***
Master's Student of the Department of Information and Computer Technologies
[2] ***I. Makhovych***
PhD, Associate Professor of the Department of Philology and Translation
[1,2] *Kyiv National University of Technologies and Design, Kyiv*

METHODOLOGY FOR AUTOMATED REST API SECURITY ANALYSIS WITHIN CONTINUOUS INTEGRATION AND CONTINUOUS DEPLOYMENT (CI/CD) PIPELINES

REST APIs are widely used for data exchange and business logic execution in cloud-native microservices. Agile methodologies and DevOps workflows allow development teams to deploy code changes frequently. Traditional security evaluations, such as periodic manual penetration testing or post-deployment vulnerability scans, can be difficult to reconcile with this delivery speed. Integrating automated checks into development can provide earlier feedback while complementing manual security assessment [1].

APIs expose application functionality, object identifiers, and data structures that require appropriate access controls. Relevant risks include Broken Object Level Authorization (BOLA), Broken Authentication, and Mass Assignment. In the OWASP API Security Top 10 2023, Mass Assignment is addressed within Broken Object Property Level Authorization [2]. To mitigate these risks, security testing can “shift left” by integrating security controls into the automated build and test steps of the CI/CD workflow.

The proposed methodology establishes a multi-layered security gate within the CI/CD workflow, for example in GitHub Actions, GitLab CI, or Jenkins. It comprises three phases: static analysis and schema validation, dynamic API fuzzing and vulnerability scanning, and finding evaluation with automated gatekeeping. During the initial phase, static analysis and schema linting examine the codebase and API definitions at the build stage required by the selected tools. Contract validation tools such as Spectral check OpenAPI descriptions against rules for security scheme declarations, request and response schemas, and HTTPS server URLs. These checks validate the description rather than prove that the running application enforces the declared controls. Static application security testing (SAST) and secret scanning can identify code weaknesses and exposed credentials. Framework-specific rules can flag authorization weaknesses.

After static validation, dynamic testing is performed in an ephemeral, containerized environment. The OpenAPI specification provides endpoint, method, and payload information for dynamic security scanners such as ZAP and property-based API testing tools such as Schemathesis [3]. Selected scanning rules and fuzzing strategies can test input handling and potential injection vulnerabilities. BOLA checks require test identities with different access permissions, objects with known ownership, and assertions about expected authorization outcomes [2]. Controlled request bursts can test configured rate limits and expected HTTP 429 responses. These checks address selected security properties rather than guarantee complete coverage of the OWASP API Security Top 10.

In the final phase, findings from static and dynamic checks are aggregated and reviewed against a defined deployment policy. The Common Vulnerability Scoring System (CVSS) describes vulnerability severity; its Base score alone is not a complete measure of organizational risk [4]. The following simplified gate uses a threshold of 7.0 for validated, unresolved vulnerabilities scored using a consistent CVSS version and metric group:

$$Gate\ Decision = \begin{cases} FAIL\ (Block\ Build), if\ \exists\ Vulnerability\ with\ CVSS\ \geq 7.0 \\ PASS\ (Proceed), if\ \forall\ Vulnerabilities\ have\ CVSS\ < 7.0 \end{cases} \quad (1)$$

If any vulnerability meets or exceeds the predefined CVSS threshold (≥ 7.0), the engine automatically triggers a build failure, blocking deployment and delivering immediate feedback to the developer. Otherwise, the build passes the security gate and proceeds seamlessly through the deployment pipeline. This threshold is a project-specific policy choice. Automatic approval also requires successful completion of the mandatory checks and triage of relevant findings; passing the gate does not guarantee the absence of vulnerabilities.

Two strategies are proposed to limit the effect of security testing on pipeline execution time:

Differential Dynamic Scanning: Prioritize changed controllers and affected endpoints during pull requests. A git diff identifies code changes, while change-impact analysis maps them to dependent endpoints. Changes to shared authorization logic, middleware, or configuration require broader regression testing.

Tiered Pipeline Execution: Run schema linting, fast static checks, and targeted endpoint tests during pull-request builds, with a target duration below five minutes. Schedule broader dynamic testing nightly and for release candidates; checks designated as release requirements must complete before deployment.

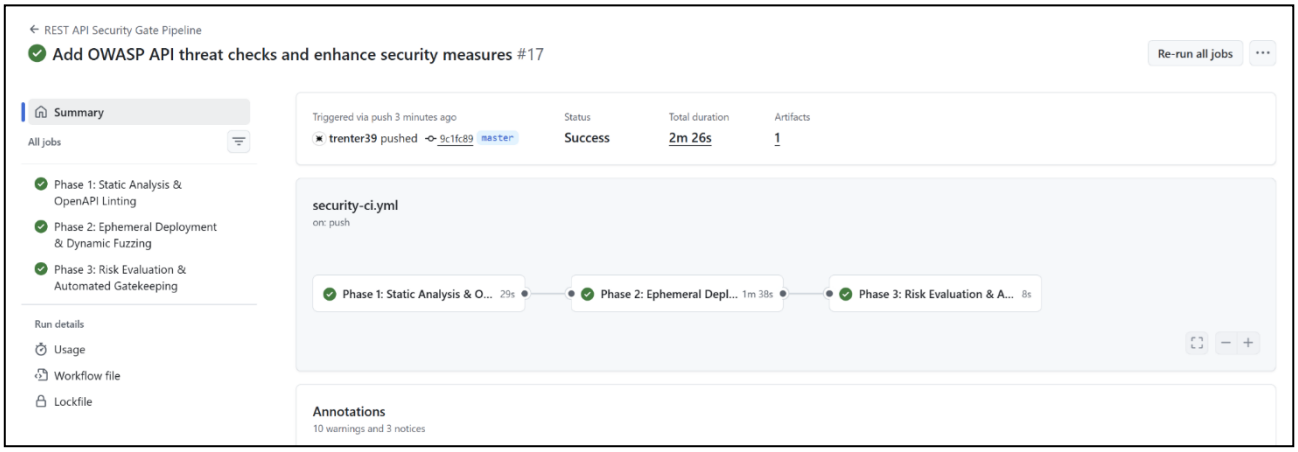


Figure 1 – Execution flow and execution time metrics of the automated REST API security pipeline in GitHub Actions. The workflow run shown represents a preliminary evaluation performed by the author

Figure 1 illustrates the three-phase workflow in GitHub Actions. The displayed execution, run #17, was triggered by a push to the master branch and completed with a Success status. Table 1 summarizes the execution times shown for this individual run.

Table 1. Execution times displayed for the GitHub Actions workflow in Figure 1

Workflow stage or metric	Displayed duration
Phase 1: Static Analysis & OpenAPI Linting	29 s
Phase 2: Ephemeral Deployment & Dynamic Fuzzing	1 min 38 s
Phase 3: Risk Evaluation & Automated Gatekeeping	8 s
Total workflow duration	2 min 26 s

By delivering security feedback directly within pull requests, the proposed workflow helps mitigate remediation effort through early vulnerability detection.

The proposed methodology combines static analysis, OpenAPI-driven dynamic testing, and automated security gatekeeping within CI/CD pipelines. A preliminary evaluation demonstrates the feasibility of integrating automated API security checks into the software delivery process while maintaining acceptable pipeline execution times. This approach enables earlier identification of potential security issues before deployment and supports the implementation of DevSecOps practices. Future work will focus on validating the methodology across multiple applications and investigating machine-learning-assisted generation of stateful API test scenarios.

References

1. Rajapakse R. N., Zahedi M., Babar M. A., Shen H. Challenges and solutions when adopting DevSecOps: A systematic review. Information and Software Technology. 2022. Vol. 141. Article 106700. https://doi.org/10.1016/j.infsof.2021.106700

2. OWASP Foundation. OWASP Top 10 API Security Risks – 2023. https://owasp.org/API-Security/editions/2023/en/0x11-t10/ (accessed 16 October 2026).

3. Golmohammadi A., Zhang M., Arcuri A. Testing RESTful APIs: A Survey. ACM Transactions on Software Engineering and Methodology. 2023. Vol. 33, No. 1. Article 27. P. 1–41. https://doi.org/10.1145/3617175

4. FIRST. Common Vulnerability Scoring System v4.0: User Guide. https://www.first.org/cvss/v4.0/user-guide (accessed 17 October 2026).