Universe International Journal of Interdisciplinary Research
(International Peer Reviewed Refereed Journal)
DOI No. – 08.2020-25662434

© UIJIR | ISSN (O) – 2582-6417
NOV 2025 | Vol. 06 Issue 06
www.uijir.com

INTEGRATING AUTOMATED DEVSECOPS SECURITY SCANNING INTO OPEN-SOURCE SOFTWARE: A MULTI-TOOL ANALYSIS USING HEALTHCHECKS AND DOKEMON

Author's Name: Akor Jacob Terungwa[1]

Affiliation:

1. Department of Security and Network Engineering, Innopolis University, Russia.

Corresponding Author Name & Email Id: Akor Jacob Terungwa, akorjacob54@gmail.com

ABSTRACT

The increasing adoption of DevSecOps has redefined software development by embedding security practices throughout the continuous integration and delivery (CI/CD) lifecycle. This research focuses on implementing and evaluating an automated DevSecOps security framework that integrates multiple open-source tools including SonarQube, Bandit, CodeQL, Semgrep, and Gitleaks to enhance software assurance in open-source projects. Using Healthchecks and Dokemon as case studies, the project systematically analyzed source code, dependencies, and configuration vulnerabilities within GitHub Actions pipelines. Each tool contributed a unique layer of analysis: SonarQube for quality and maintainability, Bandit for Python security checks, Semgrep for pattern-based vulnerability scanning, CodeQL for semantic analysis, and Gitleaks for secret detection. Results demonstrated that the combined DevSecOps pipeline effectively identified vulnerabilities early in the development process, improved code reliability, and reduced manual intervention in security auditing. Comparative findings across both projects highlighted that integrated automation yields higher vulnerability coverage and better compliance with secure coding practices. Overall, the study validates that a well-orchestrated DevSecOps pipeline not only strengthens software security posture but also promotes continuous security validation in open-source development ecosystems.

Keywords: Integrating Automated Devsecops, multi-Tool Analysis , Healthchecks And Dokemon

DOI Link :: https://doi-ds.org/doilink/12.2025-59926622/UIJIR

© UIJIR | ISSN (O) – 2582-6417
NOV 2025 | Vol. 06 Issue 06
www.uijir.com

INTRODUCTION

In today's software ecosystem, automation, containerization, and distributed workloads have increased the reliance on open-source tools for system scheduling, orchestration, and operational monitoring. Two such tools namely; **Healthchecks** and **Dokemon** play important roles in modern deployment environments. **Healthchecks** is an open-source service used to monitor scheduled jobs and alert users when these jobs fail or exceed expected execution windows. Its reliability is crucial in CI/CD automation pipelines where cron jobs, backups, and system tasks must execute predictably. **Dokemon**, on the other hand, is an open-source web-based dashboard designed for managing Docker containers, monitoring running services, and performing container-level operations in a user-friendly interface. Because Dokemon interacts directly with container runtimes and system-level resources, its security posture is essential for safeguarding compute environments.

Given the operational importance of both tools, ensuring their reliability, maintainability, and security is paramount. This report performs a comprehensive DevSecOps analysis of **Healthchecks and Dokemon**, incorporating an integrated toolchain including SonarQube, coverage.py, Bandit, CodeQL, Gitleaks, and Semgrep across all CI/CD stages from commit to deployment. Recent studies show that combining static analysis, dependency auditing, and multi-tool DevSecOps pipelines significantly enhances vulnerability detection and overall system robustness (**Rahman & Williams, 2022; Yeboah & Popoola, 2024**). Additionally, research confirms that orchestrating multiple scanning tools within an automated pipeline provides more reliable and scalable security assurance than manual reviews (**Shah & Dubey, 2021**).

Together, Healthchecks and Dokemon present an ideal opportunity to demonstrate how integrated DevSecOps practices can strengthen the quality and security of open-source software projects.

Problem Statement

Despite the increasing adoption of DevSecOps practices, many open-source projects still lack automated and fully integrated security validation within their development pipelines. In tools such as **Healthchecks** and **Dokemon**, security scanning, dependency auditing, and code-quality checks are often performed manually or inconsistently, allowing vulnerabilities and weak configurations to go undetected. Research shows that without automated multi-tool pipelines, software projects face reduced vulnerability coverage and increased technical debt (**Rahman & Williams, 2022; Shah & Dubey, 2021**).

DOI Link :: https://doi-ds.org/doilink/12.2025-59926622/UIJIR

© UIJIR | ISSN (O) – 2582-6417
NOV 2025 | Vol. 06 Issue 06
www.uijir.com

This study therefore addresses the challenge of **how to design and evaluate an automated DevSecOps pipeline that integrates multiple security tools to improve code reliability, vulnerability detection, and overall software assurance** in open-source environments.

Objectives of the Study

This study aims to:

- **Evaluate** the effectiveness of DevSecOps tools (SonarQube, Bandit, Semgrep, CodeQL, Gitleaks, coverage.py) for detecting code weaknesses and security flaws across two open-source systems, Healthchecks and Dokemon.
- **Automate** static analysis, dependency scanning, secret detection, and quality assessment as part of a continuous integration workflow.
- **Compare** vulnerability severity, code quality, dependency risks, and test coverage across both projects.
- **Enhance** the software assurance process by demonstrating how integrated DevSecOps pipelines can support secure, maintainable, and reliable open-source development.

METHODOLOGY

2.1 Tools & Technologies

The DevSecOps pipeline integrates:

- **SonarQube:** Unified dashboard for quality, reliability, maintainability, and security metrics.
- **coverage.py:** Test coverage measurement with Cobertura XML reporting.
- **Bandit:** Python-specific static security analysis.
- **Semgrep:** Rule-based static analysis covering Python, JavaScript, and Docker logic.
- **CodeQL:** Semantic code analysis for injection paths, tainted data flows, and complex logic issues.
- **Gitleaks:** Detection of hardcoded secrets.
- **OWASP Dependency-Check:** CVE-based dependency scanning for Dokemon.
- **GitHub Actions:** CI/CD automation orchestrating all tools.

2.2 CI/CD Workflow

Each commit triggers automated:

DOI Link :: https://doi-ds.org/doilink/12.2025-59926622/UIJIR

© UIJIR | ISSN (O) – 2582-6417
NOV 2025 | Vol. 06 Issue 06
www.uijir.com

- **Test execution + coverage reporting**
- **Static code security scanning (Bandit, CodeQL, Semgrep)**
- **Secret scanning (Gitleaks)**
- **Dependency vulnerability scanning**
- **Result aggregation in SonarQube & GitHub Actions annotations**

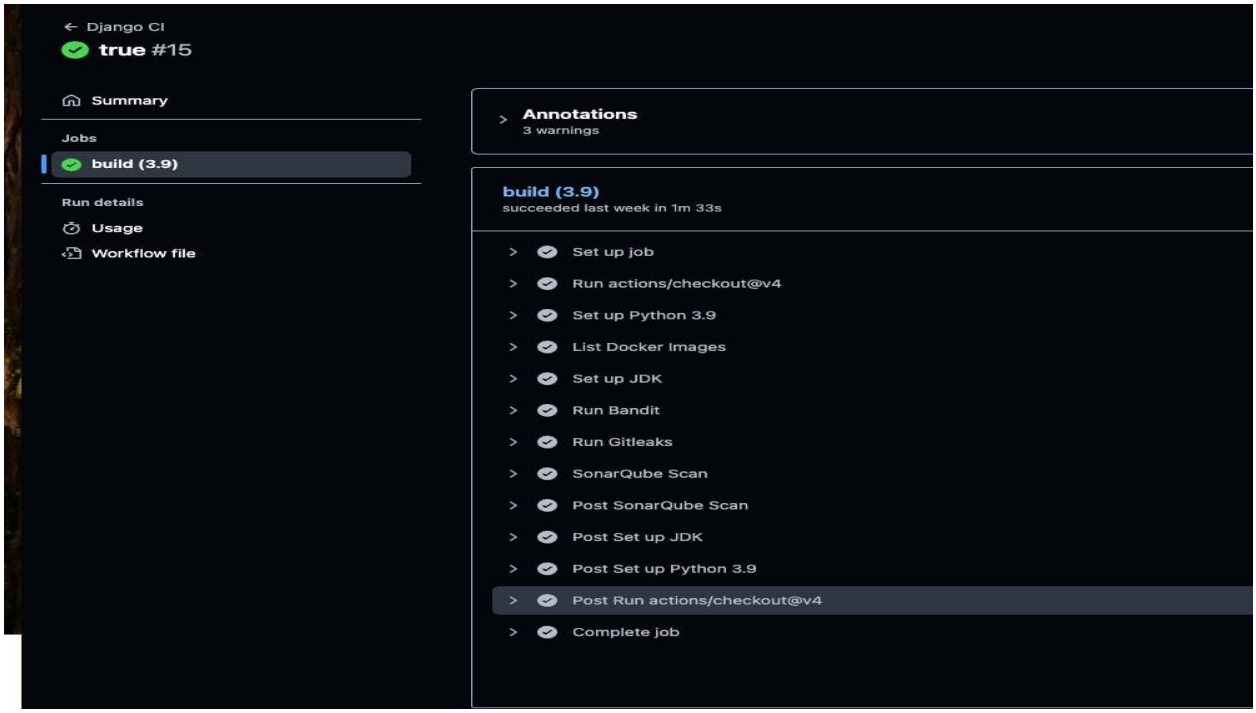

Figure 1: CI/CD pipeline overview integrating coverage.py, Bandit, CodeQL, Semgrep, and Gitleaks via GitHub Actions.

3. Results

3.1 SonarQube Quality & Security Summary (Healthchecks)

SonarQube revealed:

- **92% test coverage**, but unevenly distributed across modules
- **942 vulnerabilities** and **363 security hotspots**, mostly unreviewed
- **711 code smells** with moderate technical debt
- **1.9% code duplication**
- High-risk files clustered around complex views and API handlers

DOI Link :: https://doi-ds.org/doilink/12.2025-59926622/UIJIR

Universe International Journal of Interdisciplinary Research
(International Peer Reviewed Refereed Journal)
DOI No. – 08.2020-25662434

© UIJIR | ISSN (O) – 2582-6417
NOV 2025 | Vol. 06 Issue 06
www.uijir.com

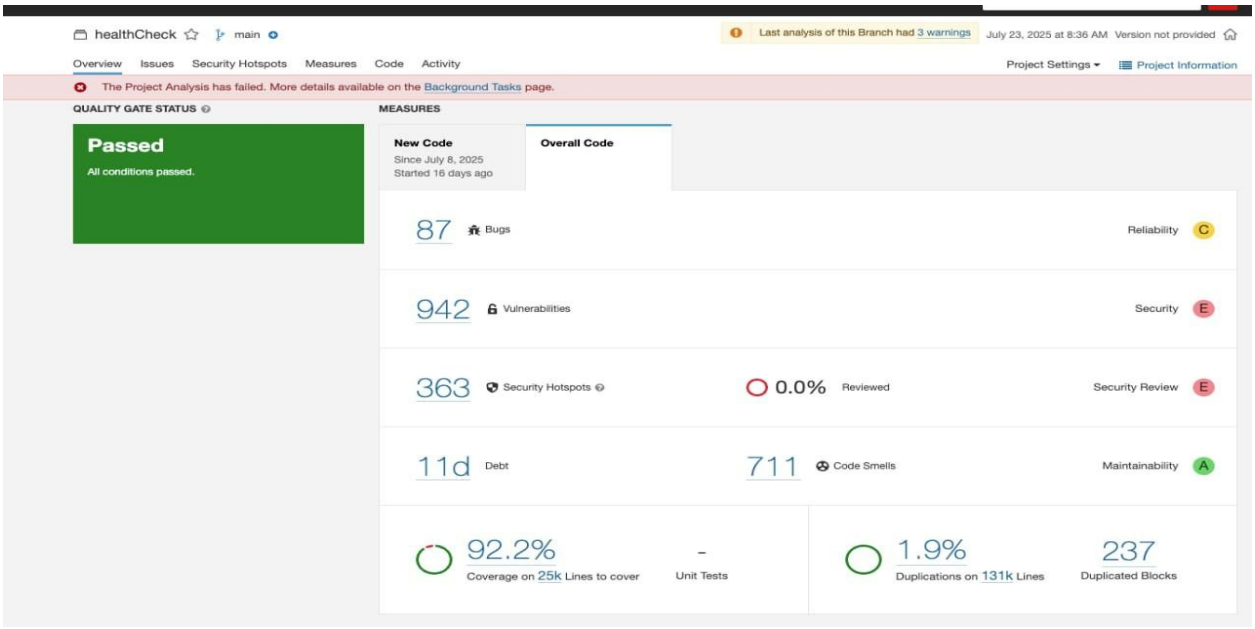

Figure 2: SonarQube dashboard summarizing code quality, vulnerabilities, and coverage

3.2 Test Coverage Findings

coverage.py and SonarQube revealed:

- Several modules with **0–20% coverage**, especially around API integrations
- High-complexity files lacking tests
- Coverage strong in utility and core scheduling logic

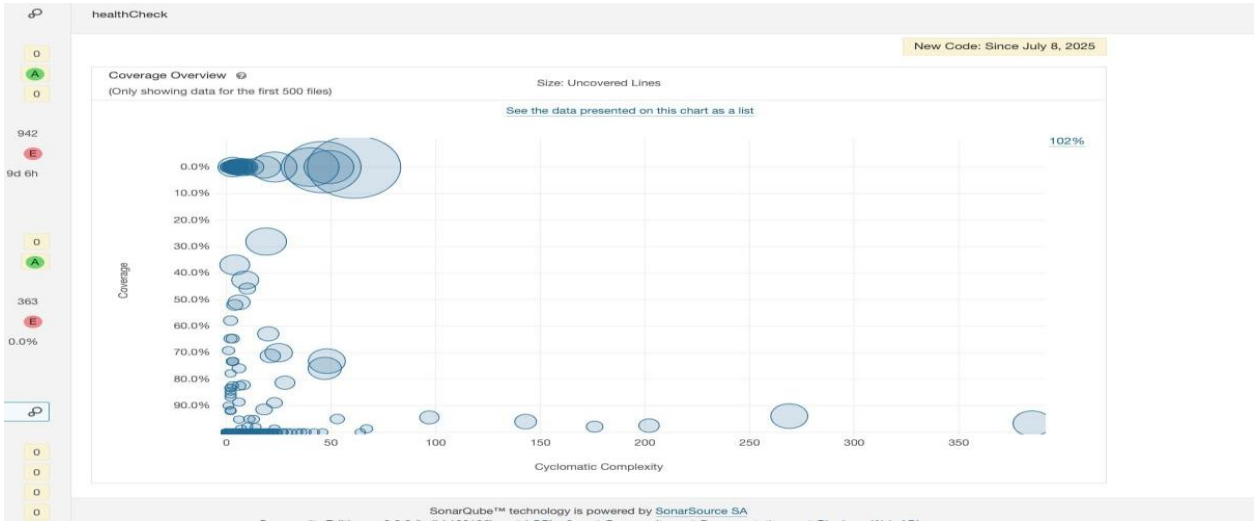

Figure 3: Module-Level Test Coverage Visualization

DOI Link :: https://doi-ds.org/doilink/12.2025-59926622/UIJIR

© UIJIR | ISSN (O) – 2582-6417
NOV 2025 | Vol. 06 Issue 06
www.uijir.com

3.3 Security Scanning Findings

Bandit

Common flagged issues:

- Unsafe functions (e.g., eval())
- Weak temporary file handling
- Missing input validation

CodeQL

Identified:

- Tainted variables reaching unsafe sinks
- Insecure shell commands
- Authentication logic inconsistencies
- Data-flow paths leading to potential injection vulnerabilities

Semgrep

Detected:

- JavaScript XSS-prone patterns (document.write, innerHTML)
- Missing CSRF enforcement in Django views (@csrf_exempt)
- Insecure Docker socket access
- Hardcoded configuration values

Dependency-Check (Dokemon)

- No high-severity CVEs, but several outdated packages flagged
- All dependencies permissively licensed (MIT, BSD, Apache-2.0)

Gitleaks

- Detection of committed test secrets
- Exposed environment-style tokens requiring immediate rotation

DOI Link :: https://doi-ds.org/doilink/12.2025-59926622/UIJIR

Universe International Journal of Interdisciplinary Research
(International Peer Reviewed Refereed Journal)
DOI No. – 08.2020-25662434

© UIJIR | ISSN (O) – 2582-6417
NOV 2025 | Vol. 06 Issue 06
www.uijir.com

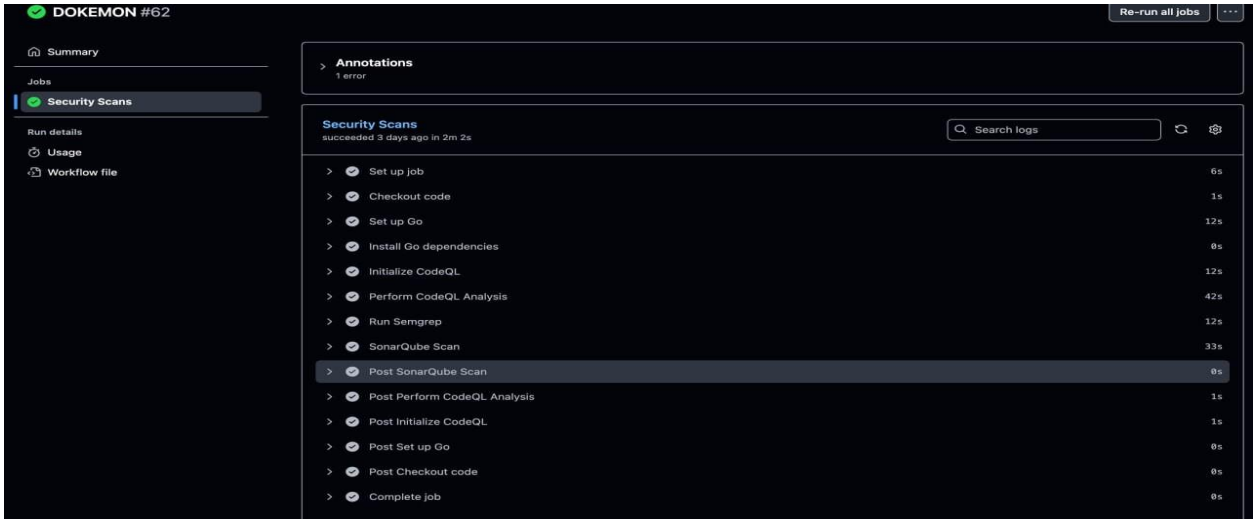

Figure 4: Security Scan Overview – Bandit, Semgrep, CodeQL, Gitleaks

3.4 Pipeline Observations

GitHub Actions provided:

- Unified feedback across all scanning stages
- Inline annotations linking findings directly to code
- Reproducible pipelines ensuring consistent scan execution

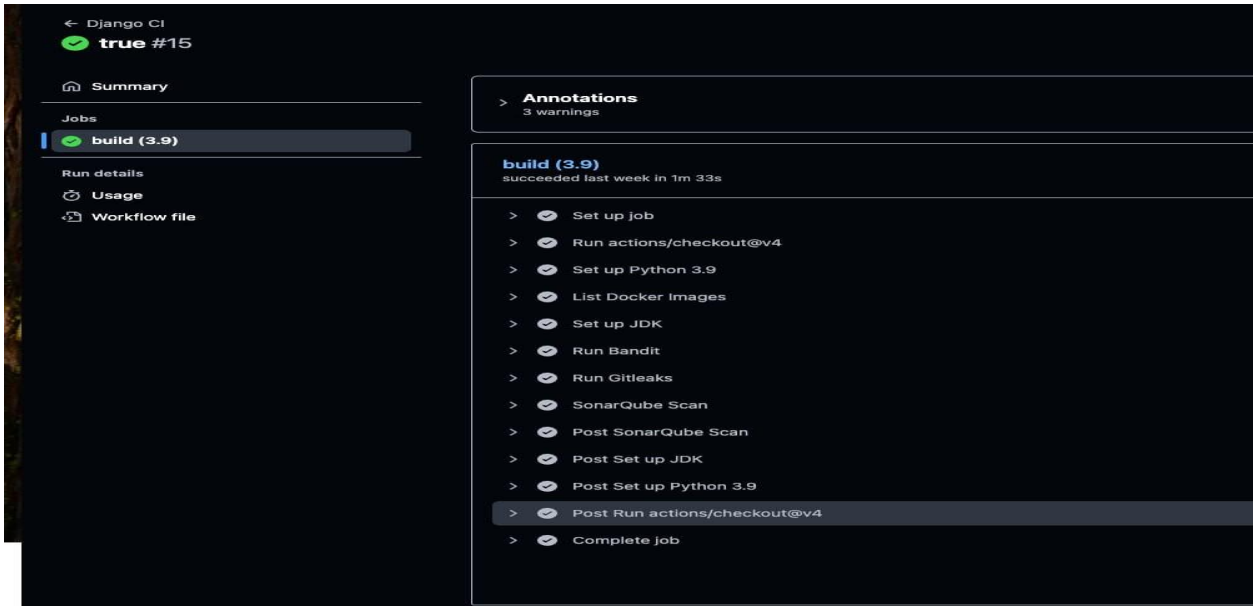

Figure 5: GitHub Actions Workflow Summary – Security & Quality Stages

DOI Link :: https://doi-ds.org/doilink/12.2025-59926622/UIJIR

© UIJIR | ISSN (O) – 2582-6417
NOV 2025 | Vol. 06 Issue 06
www.uijir.com

DISCUSSION

4.1 Code Quality & Maintainability

The SonarQube dashboard demonstrates that Healthchecks and Dokemon both maintain generally clean, maintainable codebases with manageable technical debt. However, low-coverage areas and complex modules require targeted testing and refactoring.

4.2 Security Posture

Findings reveal serious risks:

- **High vulnerability counts** in Healthchecks
- **Semantic flaws** in Dokemon detected via CodeQL
- **XSS and CSRF exposure** from Semgrep rules
- **Secret leakage** through Gitleaks requiring rotation

These underscore the need for earlier and stricter enforcement of security policies.

4.3 Limitations

- False positives from Semgrep and Gitleaks
- Absence of dynamic (DAST) scanning
- No runtime container behavior analysis
- Limited tests for user interfaces and multi-node orchestration flows

CONCLUSION

5.1 Summary of Findings

Integrating SonarQube, Bandit, CodeQL, Semgrep, Gitleaks, and Dependency-Check into GitHub Actions significantly improved visibility into code quality and security. Both Healthchecks and Dokemon showed strong foundational quality but also exhibited critical vulnerabilities, dependency inconsistencies, and authentication/authorization defects.

5.2 Recommendations

- Expand automated test suites to cover high-complexity modules
- Enable DAST tools (OWASP ZAP, Nikto) for runtime analysis

DOI Link :: https://doi-ds.org/doilink/12.2025-59926622/UIJIR

- Adopt container-aware scanning (Trivy, Grype) for Dokemon
- Enforce strict secret-management practices (Vault, SOPS)
- Continuously update Semgrep and CodeQL rulesets

5.3 Future Work

- Integrate eBPF-based runtime monitoring
- Add API fuzz testing
- Strengthen supply-chain security with SBOM generation (CycloneDX, Syft)

REFERENCES

1. Rahman, F., & Williams, L. (2022). *Software security in DevSecOps pipelines: A systematic review*. *Journal of Systems and Software*.
2. Shah, V., & Dubey, S. (2021). *Leveraging automation for secure CI/CD pipelines*. *IEEE Software, 38*(5), 65–72.
3. Yeboah, D., & Popoola, S. (2024). *Enhancing software reliability through integrated DevSecOps toolchains*. *International Journal of Secure Software Engineering, 12*(1), 44–59.
4. Fitzgerald, B., & Stol, K. J. (2017). *Continuous software engineering: A roadmap and agenda*. *Journal of Systems and Software, 123*, 176–189.
5. Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps Handbook*. IT Revolution Press.
6. Williams, J., & Wichers, D. (2023). *OWASP Top Ten Web Application Security Risks*. OWASP Foundation.
7. CISA. (2023). *Software Supply Chain Security Guidance*. Cybersecurity and Infrastructure Security Agency.
8. Healthchecks. (n.d.). *Healthchecks Documentation*. https://healthchecks.io/docs
9. Dokemon. (n.d.). *Dokemon GitHub Repository*. GitHub. https://github.com
10. SonarSource. (2024). *SonarQube Documentation*. https://docs.sonarsource.com
11. Coverage.py. (2024). *Coverage.py Documentation*. https://coverage.readthedocs.io
12. Python Code Quality Authority. (2024). *Bandit Documentation*. https://bandit.readthedocs.io
13. Semgrep. (2024). *Semgrep Official Documentation*. https://semgrep.dev/docs
14. GitHub Security Lab. (2024). *CodeQL Documentation*. https://codeql.github.com/docs
15. Gitleaks. (2024). *Gitleaks Documentation*. https://gitleaks.io

DOI Link :: https://doi-ds.org/doilink/12.2025-59926622/UIJIR

© UIJIR | ISSN (O) – 2582-6417
NOV 2025 | Vol. 06 Issue 06
www.uijir.com

16. OWASP Foundation. (2023). *OWASP Dependency-Check*. https://owasp.org/www-project-dependency-check/

17. Docker Inc. (2024). *Docker Documentation*. https://docs.docker.com

DOI Link :: https://doi-ds.org/doilink/12.2025-59926622/UIJIR