Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 1, 393-402
2025
Publisher: Learning Gate
DOI: 10.55214/25768484.v9i1.4153

Securing the development and delivery of modern applications

Matej Bašić[1*], Zlatan Morić[2], Jasmin Redžepagić[3], Josip Torbar[4]
[1,2,3,4]Department of System Engineering and Cybersecurity, Algebra University Zagreb, Croatia; mbasic2@algebra.hr (M.B.) zlatan.moric@algebra.hr (Z.M.) jasmin.redzepagic@algebra.hr (J.R.) jtorbar@algebra.hr (J.T.)

Abstract: This study analyzes the significant difficulties and solutions for ensuring the security of developing and deploying contemporary software applications in the current fast-paced technological landscape. It examines the escalating hazards linked to expedited software delivery cycles via agile and DevOps methodologies, emphasizing critical domains such as static code analysis, CI/CD pipeline security, container image scanning, and container platform security. This research consolidates findings from existing studies and practical case scenarios, assessing the efficacy of SonarQube, Clair, Trivy, and Kube-bench in detecting vulnerabilities, improving operational efficiency, and guaranteeing adherence to industry standards. Case studies, such as Skyscanner's implementation of SonarQube and DAZN's utilization of Checkmarx, underscore the concrete advantages of incorporating sophisticated security protocols. The results underscore the significance of multi-tiered security approaches and highlight prospects for utilizing artificial intelligence to improve vulnerability identification and real-time surveillance. This paper offers practical recommendations to assist organizations in tackling current security issues and anticipating future threats.

Keywords: *Application security, Container image scanning, Container platform security, Continuous Integration/Continuous Delivery, DevSecOps, vulnerability detection, Software Development Lifecycle, Static Code Analysis,*

1. Introduction

The development and delivery of modern software applications have become increasingly complex. As organizations adopt agile and DevOps methodologies to accelerate delivery cycles, they face heightened security challenges across all stages of the Software Development Lifecycle (SDLC). The need to detect vulnerabilities, secure software supply chains, and protect runtime environments has become highly important. The complexity of contemporary applications and an ever-expanding attack surface highlight the importance of integrating robust security measures seamlessly into development and operational workflows. High-profile incidents, such as the SolarWinds supply chain attack and the CodeCov breach, have further illuminated the risks inherent in insecure pipelines and dependencies, prompting a call for more rigorous approaches to application security. This paper examines critical areas in securing the development and delivery of modern applications, focusing on Static Code Analysis (SCA), Continuous Integration/Continuous Delivery (CI/CD) pipeline security, container image scanning, and container platform security. The discussion synthesizes recent research and case study findings, presenting a comprehensive view of tools and methodologies to address vulnerabilities at each application lifecycle stage. By exploring advancements such as machine learning-enhanced SCA tools, automated security scanning in CI/CD pipelines, and runtime monitoring tools for container orchestration platforms like Kubernetes, the paper highlights the critical interplay between technology, process, and governance in achieving robust security. This work contributes to the field by providing an in-depth analysis of current best practices, tools, and real-world implementations in application security. It evaluates state-of-the-art solutions, such as SonarQube, Clair, Trivy, and Kube-bench, which are

History: Received: 25 November 2024; Revised: 23 December 2024; Accepted: 1 January 2025; Published: 9 January 2025
* Correspondence: mbasic2@algebra.hr

instrumental in mitigating risks and ensuring compliance. Furthermore, this paper identifies gaps in existing practices. It outlines opportunities for future research, such as leveraging artificial intelligence for proactive vulnerability detection and implementing real-time anomaly detection in dynamic environments. The insights presented here aim to guide developers, security professionals, and organizations in adopting a proactive approach to securing the software lifecycle.

The rest of this paper is organized as follows. The next section reviews foundational and recent work that contextualizes the challenges and advancements in application security. This is followed by analyzing key security practices, including Static Code Analysis, CI/CD pipeline security, and containerized environment protection. The discussion concludes with a forward-looking perspective on future research opportunities and a summary of critical takeaways.

2. Related Work

The increasing complexity of software systems has necessitated advancements in techniques for identifying vulnerabilities and ensuring security throughout the Software Development Lifecycle. Static Code Analysis is a prominent approach for identifying vulnerabilities within non-executing source code. Despite their limitations, recent studies emphasize the utility of SCA tools in detecting security flaws. For instance, Ruiz, et al. [1] highlight the transformative role of artificial intelligence in enhancing static code analysis, paving the way for more sophisticated tools. Similarly, Lipp, et al. [2] show that combining results from multiple analyzers can reduce false negatives, although vulnerabilities remain undetected in real-world applications. The importance of SCA tools is further validated by Alqaradaghi and Kozsik [3] who note that no single tool can comprehensively detect vulnerabilities, emphasizing the need for integration and tool diversity in software testing. The security implications of CI/CD pipelines have gained significant attention due to their critical role in modern DevOps workflows. As Pan [4] detail, CI/CD pipelines are susceptible to various attack vectors, such as malicious code injection and outdated dependencies, leading to severe consequences. Paule, et al. [5] identify 22 vulnerabilities in industry pipelines, highlighting the need for enhanced threat modeling. Ho-Dac and Vo [6] propose integrating open-source security tools into pipelines, aligning with the "shift-left" security philosophy emphasizing early vulnerability detection. Meanwhile, Marandi, et al. [7] highlight the importance of automating security scanning using dynamic and static analysis tools to mitigate risks in DevSecOps environments.

Dakic, et al. [8] highlight that because modern application development has redefined how teams develop their solutions, CI/CD tools need advanced privileges; they have to be able to access code repositories, etc., creating a big attack surface. Furthermore, Dakic, et al. [9] also note that, without any extra configuration, only a limited number of messages pass through the audit system, mainly authentication/authorization and SELinux messages. This creates a more extensive set of problems, as auditing is a fundamental aspect of security that must be adequately implemented and monitored for application security to be high enough. There are also problems with vulnerabilities in system architecture, configurations, and operation practices that need to be addressed to safeguard sensitive data [10]. Container platform security has also emerged as a focal area, with platforms like Docker and Kubernetes at the forefront. Bhardwaj, et al. [11] stress the need for automated vulnerability detection and compliance enforcement in containerized ecosystems. Shevchuk, et al. [12] explore Kubernetes-specific vulnerabilities, advocating for best practices to secure CI/CD processes. Afifah, et al. [13] propose using code obfuscation techniques within CI/CD pipelines to safeguard against unauthorized access, demonstrating the growing intersection of security and automation in containerized workflows.

Dynamic security testing tools integrated into CI/CD pipelines have shown promise in addressing runtime vulnerabilities. Rangnau, et al. [14] present a case study integrating dynamic testing techniques, revealing challenges and opportunities in the DevSecOps paradigm. Additionally, Ponaka [15] illustrates the effectiveness of security gates in preventing the deployment of critical vulnerabilities, advocating for automated checks throughout the SDLC.

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 1: 393-402, 2025
DOI: 10.55214/25768484.v9i1.4153

Recent advancements in threat modeling and policy enforcement have strengthened CI/CD security frameworks. Nikolov and Aleksieva-Petrova [16] propose a Jenkins-based framework for embedding threat modeling into pipelines. Morales and Yasar [17] emphasize cultural and technical barriers to implementing secure pipelines in regulated environments. These studies highlight the necessity of a holistic approach combining technical tools, governance, and cultural shifts. Recent research shows the role of advanced tools and methodologies in mitigating CI/CD and container vulnerabilities. Singh [18], Bajpai and Lewis [19] discuss best practices for secure CI/CD implementations, emphasizing the importance of collaboration between development and operations teams. Meanwhile, Vassallo, et al. [20] highlight the potential of automated linters to detect misconfigurations in CI/CD pipelines, demonstrating the utility of semantic analysis tools.

3. Static Code Analysis

Static Code Analysis, also called Source Code Analysis, is an integral part of the Code Review process, often categorized under white-box testing [21]. It is typically conducted during the Security Development Lifecycle (SDL) implementation phase. This method uses Static Code Analysis tools to identify potential vulnerabilities in non-running (static) source code by employing techniques such as Taint Analysis and Data Flow Analysis. Beyond enhancing application security, static code analysis also improves the overall quality and performance of the code [2]. This analysis can be performed manually or automatically, with each approach offering distinct advantages.

Manual analysis involves developers reviewing the source code to detect vulnerabilities that automated tools might overlook [1]. This method heavily relies on the developer's expertise and ability to recognize potential security issues by examining the code. However, it can be inefficient for analyzing large codebases. On the other hand, automatic static code analysis leverages specialized tools to scan the entire codebase for known vulnerabilities. These tools use sophisticated techniques, including Taint Analysis and Data Flow Analysis, to identify potential risks efficiently [22].

3.1. Data Flow Analysis

Data Flow Analysis is a critical technique in static code analysis that tracks data movement through software from input points to output points [3]. This process uses a control flow graph, an abstract representation of the program or procedure. In the graph, each node represents a basic block, and directed edges illustrate the control flow jumps between blocks. Blocks without incoming edges are designated as entry blocks, while those without outgoing edges are exit blocks. For instance, Figure 1 illustrates a simple Control Flow Graph, where "NODE 1" is the entry block, and "NODE 6" is the exit block.

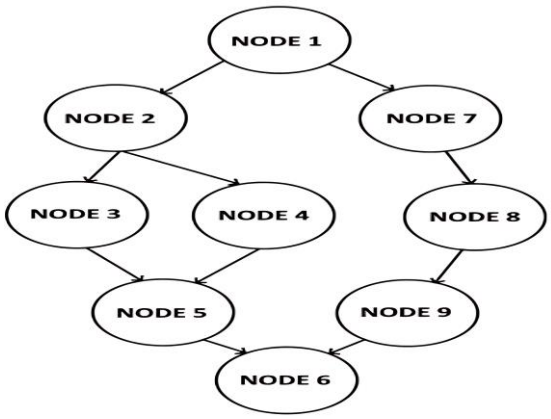

Figure 1.
Example Control Flow Graph where NODE 1 represents the entry block, and NODE 6 represents the exit block.

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 1: 393-402, 2025
DOI: 10.55214/25768484.v9i1.4153

This approach helps uncover vulnerabilities caused by improper data validation, storage, or handling, such as buffer overflows or insecure data processing [23]. By tracing data movements, developers can locate points where sensitive information might be exposed to unauthorized access or modifications.

3.2. Taint Analysis

Taint Analysis tracks potentially harmful (tainted) data flow through an application [24]. It identifies input sources susceptible to manipulation, such as form fields, URL parameters, or HTTP headers. The analysis then monitors how this data propagates within the application, identifying cases where it reaches sensitive processing points without proper sanitization or validation. This helps prevent attacks like SQL injection, cross-site scripting (XSS), and other injection-based vulnerabilities [25].

3.3. Increasing Application Vulnerabilities and Available Tools

The paper by Efimov, et al. [25] explicitly mentions the count of vulnerabilities rising in early 2023. Specifically, it states that information obtained from the CVE vulnerability database reported 7,015 identified vulnerabilities in the first quarter of 2023 [26]. Various automatic static code analysis tools are available, including popular options like SonarQube, Checkmarx, and Micro Focus Fortify Static Code Analyzer. These tools streamline the detection of vulnerabilities, improve code quality, and align with security best practices.

For example, Skyscanner, a global travel marketplace catering to over 100 million users monthly, adopted SonarQube to maintain consistent development practices [27]. This tool helped Skyscanner achieve immediate success by enhancing code quality, facilitating developer communication, and ensuring timely delivery across time zones. Similarly, DAZN, an international sports streaming service, utilized Checkmarx SAST and Codebashing solutions to secure its codebase [28]. By integrating CxSAST and CxSCA, DAZN's developers and security teams prioritized critical open-source vulnerabilities, saved time, and enhanced their application security testing strategies.

3.4. Tailoring Solutions to Organizational Needs

While many tools exist, there is no universal solution. Organizations must evaluate and select tools aligning with their requirements, development workflows, and security objectives. This tailored approach ensures maximum efficacy in vulnerability detection and code quality improvements.

4. Container Image Scanning

Container image scanning is critical for ensuring the security and integrity of containerized applications [29]. A container image represents a packaged application and all its dependencies, encapsulating everything necessary to run the application in a consistent and predictable environment. These images are executable software bundles designed to operate independently, with specific and well-defined assumptions about the runtime environment.

Container images are constructed in a layered architecture, beginning with a base image, often a minimal operating system such as Alpine Linux, Ubuntu, or Red Hat Linux [30]. This base layer provides the foundational environment, while additional layers are added for specific changes, modifications, or enhancements. These extra layers commonly include software libraries, binaries, application code, configuration files, and environment variables. The layered approach simplifies the image creation process and enhances efficiency. Each layer is cached independently, meaning only the modified layers must be rebuilt and redeployed when updates are made to the application. This feature optimizes storage and accelerates deployment, as unchanged layers can be reused across multiple builds or versions. However, once a container image is built, it is crucial to ensure that it does not introduce security vulnerabilities or compliance risks into the deployment environment. Container image scanning safeguards by analyzing the image's contents for known vulnerabilities. These vulnerabilities may include outdated software libraries, misconfigured components, or other issues that could compromise the application's security. Tools such as Clair and Trivy have become popular choices for this task [31].

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 1: 393-402, 2025
DOI: 10.55214/25768484.v9i1.4153

These tools automate the vulnerability detection process and provide actionable insights for remediation. They scan the image against extensive vulnerability databases, flagging any components with known security issues.

By integrating container image scanning into the development and deployment pipeline, organizations can significantly reduce the attack surface and mitigate security risks before they reach production [8]. This approach strengthens applications' security posture and ensures compliance with organizational and industry-specific security standards.

5. Pipeline Security

A "pipeline" refers to a series of structured steps, commonly called pipeline phases, required to deliver software from development to production [18]. These phases typically include building, testing, and deploying the software, though more complex pipelines may include additional steps, such as static code analysis, infrastructure provisioning, and monitoring. These core phases often illustrate an example of a simple CI/CD pipeline, demonstrating how code transitions seamlessly from commit to production.

CI/CD pipelines automate the manual intervention historically involved in software development and deployment. By integrating automation into the build, test, and deployment phases and infrastructure provisioning, CI/CD pipelines enable development teams to implement changes automatically tested and prepared for deployment [7]. This automation fosters faster delivery cycles, improved collaboration, and more reliable production systems. However, their security is paramount as CI/CD pipelines form the backbone of modern DevOps practices. The security of the deployed code is inherently tied to the security of the CI/CD pipeline. Vulnerabilities can arise from various sources, such as incomplete or improperly configured test cases, third-party dependencies, or malicious actors compromising the pipeline. CI/CD security measures are designed to address these risks, ensuring the pipeline's integrity and the software it produces.

A reminder of the importance of CI/CD security came in December 2020 with the SolarWinds supply chain attack [4]. This sophisticated breach targeted SolarWinds' Orion platform, widely used for network management, affecting up to 18,000 organizations, including government agencies and major corporations. Threat actors infiltrated SolarWinds' CI/CD pipeline, compromising the core processes where code is tested, packaged, containerized, and signed. The attackers introduced "SunSpot" malware, which operated with high privileges and scanned for Orion builds to modify. This attack highlighted the vulnerabilities in CI/CD processes and raised alarm across the tech industry about securing supply chains and trusted software tools.

Another significant breach occurred in 2021 with CodeCov, a popular code coverage tool used by over 29,000 customers worldwide [32]. In this incident, malicious actors compromised a script provided by CodeCov, a tool developers routinely used to assess code coverage. The altered script was designed to harvest sensitive data, such as credentials and secrets, from affected CI/CD environments. Every time a developer downloaded the compromised script, it triggered malicious activity, transmitting critical data to the attackers' servers. This breach underscored the risks inherent in third-party dependencies and the importance of securing CI/CD environments against such vulnerabilities.

The configuration of CI/CD pipelines must account for their specific applications, which often involve sensitive operations. For example, when CI/CD is used to build software packages, it may require access to external resources such as Docker images or external repositories. Similarly, pipelines supporting Infrastructure as Code (IaC) often need credentials for automated deployments. In these scenarios, securely managing sensitive information is critical. Misconfigured CI/CD environments can lead to significant security breaches, emphasizing the importance of secure configurations tailored to their specific use cases.

To secure CI/CD processes effectively, organizations must invest time and resources into securing all pipeline components [19]. This includes implementing change management practices, establishing governance frameworks, and ensuring personnel are appropriately educated about the tools and

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 1: 393-402, 2025
DOI: 10.55214/25768484.v9i1.4153

technologies. Secure configurations are not automatic; they require deliberate planning and a thorough understanding of the underlying technologies. Organizations must assess their configuration and potential vulnerabilities before using any tool for security-sensitive operations like code deployment, ensuring it aligns with their security requirements and operational needs. This proactive approach is essential to mitigate risks and maintain the integrity of the CI/CD pipeline.

6. Container Platform Security

Container platform security ensures containerized applications' safe and efficient operation. Containers deployed on platforms like Docker and Kubernetes offer scalable and streamlined application management [33]. Kubernetes is a powerful option for users who prefer to customize their environment to specific needs [34]. While container runtime security has progressed, it is crucial to ensure that application code stays securely inside its container [35]. These platforms depend on container runtime, a software component responsible for executing containers. Popular container runtimes include containerd, CRI-O, Docker Engine, and Mirantis Container Runtime. Securing the underlying container platforms is critical to protecting applications and infrastructure from vulnerabilities. Despite their apparent benefits, containers need help with security and resource isolation [36].

6.1. Docker

Platform security on Docker focuses on securely configuring the Docker Daemon. This includes minimizing privileges by using user namespaces to map container users to non-root users on the host system [37]. Communication between the Docker client and Daemon should be encrypted using TLS, safeguarding against interception or tampering. Tools like Docker Bench for Security automate verifying Docker's security configuration and running checks based on the CIS Docker Benchmark. By addressing these recommendations, organizations can ensure that Docker deployments meet established security best practices.

6.2. Kubernetes

Kubernetes, a widely used container orchestration platform, requires comprehensive security measures for its various components. As Kubernetes is entirely API-driven, the primary line of defense lies in controlling access to the API [38]. Access control mechanisms should define who can interact with the cluster and what actions they can perform. TLS encryption is mandatory for all API communications within the cluster, ensuring that data exchanges remain secure. Kubernetes offers native security enhancements, such as automated policy enforcement, identity management, and monitoring capabilities, which organizations can leverage to improve governance, simplify operations, and enhance compliance.

Network security is particularly complex in Kubernetes environments due to dynamic configurations involving ports, IP addresses, and network attributes. Kubernetes natively provides network policies that define rules for pod-to-pod communication, controlling how workloads interact at the network level. However, network policies alone are insufficient to address all network security challenges. External tools are often required to secure other aspects of Kubernetes networking. Solutions like Calico and Istio extend network security capabilities, providing granular segmentation and encrypted service-to-service communication [39].

Resource management also plays a critical role in securing container platforms. By default, containers in Kubernetes run without limitations on compute resources, which could lead to performance degradation or denial-of-service conditions if a single container monopolizes resources. Kubernetes allows administrators to configure resource quotas and limits for CPU, memory, and storage, ensuring fair allocation and protecting against resource exhaustion. These quotas also help enforce best practices for efficient resource utilization across the cluster.

Privilege management is another essential aspect of container security. Containers with elevated privileges can access host resources, potentially leading to security breaches. Kubernetes can be configured to restrict privileged container usage and enforce policies that limit container capabilities,

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 1: 393-402, 2025
DOI: 10.55214/25768484.v9i1.4153

thereby reducing the attack surface. Similarly, disabling unnecessary kernel modules on Kubernetes nodes helps minimize vulnerabilities by restricting the functionality available to potential attackers.

The placement of pods within a cluster also impacts security. Kubernetes allows administrators to use node selectors, taints, and tolerations to assign sensitive workloads to dedicated nodes with stricter security configurations. Affinity and anti-affinity rules further enhance resilience by distributing workloads across multiple nodes, reducing the risk of single points of failure.

Securing Kubernetes clusters requires specialized tools to identify vulnerabilities, enforce policies, and monitor activity [40]. Kube-bench automates security checks based on the CIS Kubernetes Benchmark, ensuring compliance with best practices. Falco provides real-time monitoring by analyzing system calls within containers to detect anomalous behavior indicative of a breach. Open Policy Agent (OPA) enforces cluster-wide security policies at the API level, managing admission controls and ensuring only compliant configurations are deployed. Tools like Calico and Istio extend security to the network layer, providing segmentation and encrypted communications. Meanwhile, Kubescape scans clusters against NSA and CISA guidelines, identifying misconfigurations and vulnerabilities and ensuring clusters remain secure.

Each tool addresses specific aspects of Kubernetes security, from compliance and policy enforcement to network protection and runtime monitoring. Integrating these tools into a cohesive security strategy helps organizations establish a robust defense against potential threats.

7. Future Works

As software systems and their security requirements continue to grow in complexity, numerous opportunities remain to improve and extend the current state of research and practices in software security. Future efforts should focus on advancing tools and methodologies to address the limitations and gaps highlighted in this work.

One promising direction is the enhancement of Static Code Analysis through integrating machine learning and artificial intelligence techniques. Current SCA tools often struggle with false positives and negatives, and advanced AI models could improve their accuracy and predictive capabilities. Additionally, further exploration into hybrid approaches, combining static and dynamic analysis methods, may provide a more comprehensive solution for identifying vulnerabilities across diverse software systems. In the CI/CD pipeline security domain, future studies should investigate better methods for securing supply chains against sophisticated attacks, such as those seen in the SolarWinds and CodeCov incidents. Research into automated threat modeling within pipelines could enable earlier detection of vulnerabilities and improve resilience against emerging threats. Furthermore, developing standardized frameworks for CI/CD security best practices could help organizations adopt consistent and effective measures, particularly for integrating third-party tools and dependencies. The focus should shift toward real-time scanning and monitoring capabilities for container image scanning and platform security. Existing tools, such as Clair and Trivy, are effective at pre-deployment scanning, but the rise of ephemeral containerized environments necessitates solutions that can continuously assess vulnerabilities during runtime. Enhanced orchestration security for Kubernetes, including better network policies and more sophisticated runtime anomaly detection, could provide greater assurance against container-based attacks.

Finally, there is a need for greater emphasis on interdisciplinary research, incorporating insights from human factors and organizational behavior into the design and implementation of secure development practices. As security is as much a cultural challenge as a technical one, future works should investigate how to foster a security-first mindset across development and operations teams. This includes developing better training programs and integrating governance models that balance innovation with security.

Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 1: 393-402, 2025
DOI: 10.55214/25768484.v9i1.4153

8. Conclusion

The increasing complexity of modern software systems and the evolving threat landscape necessitate a proactive approach to security throughout the Software Development Lifecycle. This paper has explored several critical aspects of software security, including Static Code Analysis, container image scanning, CI/CD pipeline security, and container platform security. Together, these practices form a comprehensive framework for identifying and mitigating vulnerabilities, ensuring the integrity of applications, and safeguarding infrastructure.

Static Code Analysis remains a foundational tool for identifying vulnerabilities in non-executing source code. While current tools offer great insights, challenges such as false positives and limitations in coverage highlight the need for enhanced techniques, such as the integration of machine learning. Similarly, container image scanning has become indispensable for securing containerized applications. By leveraging tools like Clair and Trivy, organizations can detect and remediate vulnerabilities in their container images before they reach production environments, reducing their overall attack surface. CI/CD pipelines, an essential component of modern DevOps practices, represent both a critical enabler of rapid software delivery and a significant vector for potential security breaches. High-profile attacks, such as the SolarWinds and CodeCov incidents, highlight the importance of securing every component of these pipelines. From integrating automated security checks to implementing robust governance frameworks, organizations must ensure that CI/CD pipelines remain a strength rather than a liability.

Finally, securing container platforms like Docker and Kubernetes is vital for maintaining application and infrastructure integrity. Privilege management, network policy enforcement, and real-time monitoring are essential to mitigating risks in dynamic and distributed environments. Tools like Kube-bench, Falco, and Open Policy Agent provide organizations with the means to enforce compliance and detect threats, ensuring robust container orchestration security. Despite significant advancements in software security practices and tools, the ever-changing nature of vulnerabilities and attack vectors demands continued caution and innovation. As highlighted in this paper, integrating security into every stage of the software lifecycle, combined with tailored solutions and ongoing research, is essential for addressing current challenges and preparing for future threats.

Transparency:

The authors confirm that the manuscript is an honest, accurate, and transparent account of the study; that no vital features of the study have been omitted; and that any discrepancies from the study as planned have been explained. This study followed all ethical practices during writing.'

Copyright:

References

[1] G. A. Ruiz, S. Robledo, and H. H. Morales, "Static code analysis: A tree of science review," *Entre Ciencia e Ingeniería*, vol. 17, no. 34, pp. 9-14, 2023. https://doi.org/10.31908/19098367.2846

[2] S. Lipp, S. Banescu, and A. Pretschner, "An empirical study on the effectiveness of static C code analyzers for vulnerability detection," in *In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 544-555)*, 2022.

[3] M. Alqaradaghi and T. Kozsik, "Comprehensive evaluation of static analysis tools for their performance in finding vulnerabilities in java code," *IEEE Access*, vol. 12, pp. 55824–55842, 2024. https://doi.org/10.1109/access.2024.3389955

[4] Z. Pan, "Ambush from all sides: Understanding security threats in open-source software CI/CD pipelines," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 1, pp. 403–418, 2024. https://doi.org/10.1109/tdsc.2023.3253572

[5] C. Paule, T. F. Dullmann, and A. Van Hoorn, "Vulnerabilities in continuous delivery pipelines? A case study," presented at the IEEE International Conference on Software Architecture Companion (ICSA-C). IEEE, pp. 102–108, 2019. https://doi.org/10.1109/icsa-c.2019.00026, 2019.

[6] H. Ho-Dac and V.-L. Vo, "An approach to enhance CI/CD pipeline with open-source security tools," *European Modern Studies Journal*, vol. 8, no. 3, pp. 408–413, 2024. https://doi.org/10.59573/emsj.8(3).2024.30

[7] M. Marandi, A. Bertia, and S. Silas, "Implementing and automating security scanning to a DevSecOps CI/CD Pipeline," presented at the World Conference on Communication & Computing (WCONF). IEEE, 2023. https://doi.org/10.1109/wconf58270.2023.10235015, 2023.

[8] V. Dakic, J. Redzepagic, and M. Basic, ""CI/CD toolset security," in *DAAAM Proceedings. DAAAM International Vienna, pp. 0161–0164, 2022. https://doi.org/10.2507/33rd.daaam.proceedings.022*, 2022.

[9] V. Dakic, K. Jakobovic, and L. Zgrablic, "Linux security in physical, virtual, and cloud environments," in *DAAAM Proceedings. DAAAM International Vienna, pp. 0151–0160, 2022. https://doi.org/10.2507/33rd.daaam.proceedings.021*, 2022.

[10] K. Loncar, J. Redzepagic, and V. Dakic, "Secure coding guidelines and standards," in *DAAAM Proceedings. DAAAM International Vienna, pp. 0191–0198, 2024. https://doi.org/10.2507/35th.daaam.proceedings.025*, 2024.

[11] A. K. Bhardwaj, P. Dutta, and P. Chintale, "Securing container images through automated vulnerability detection in shift-left CI/CD Pipelines," *Babylonian Journal of Networking*, vol. 2024, pp. 162-170, 2024. https://doi.org/10.58496/bjn/2024/016

[12] R. Shevchuk, M. Karpinski, M. Kasianchuk, I. Yakymenko, A. Melnyk, and R. Tykhyi, "Software for improve the security of kubernetes-based CI/CD pipeline," presented at the 13th International Conference on Advanced Computer Information Technologies (ACIT). IEEE, pp. 420–425, 2023. https://doi.org/10.1109/acit58437.2023.10275654, 2023.

[13] A. S. Afifah, H. Kabetta, I. K. Setia Buana, and H. Setiawan, "Code obfuscation in CI/CD pipelines for enhanced DevOps security," presented at the International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD). IEEE, pp. 137–142, 2024. https://doi.org/10.1109/icoabcd63526.2024.10704536, 2024.

[14] T. Rangnau, R. V. Buijtenen, F. Fransen, and F. Turkmen, "Continuous security testing: A case study on integrating dynamic security testing tools in CI/CD pipelines," presented at the IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC). IEEE, pp. 145–154,2020. https://doi.org/10.1109/edoc49727.2020.00026, 2020.

[15] K. R. Ponaka, "Shift-left approach for vulnerability management in SDLC," *Interantional Journal of Scientific Research in Engineering and Management*, vol. 8, no. 10, pp. 1–14, 2024. https://doi.org/10.55041/ijsrem9417

[16] L. Nikolov and A. Aleksieva-Petrova, "Framework for integrating threat modeling into a devops pipeline for enhanced software development," presented at the International Conference on Software, Telecommunications and Computer Networks (SoftCOM). IEEE, pp. 1–5, 2024. https://doi.org/10.23919/softcom62040.2024.10721871, 2024.

[17] J. A. Morales and H. Yasar, "Experiences with secure pipelines in highly regulated environments," in *Proceedings of the 18th International Conference on Availability, Reliability and Security. ACM, pp. 1–9, 2023. https://doi.org/10.1145/3600160.3605466*, 2023.

[18] N. Singh, "CI/CD Pipeline for web applications," *International Journal for Research in Applied Science and Engineering Technology*, vol. 11, no. 5, pp. 5218–5226, 2023. https://doi.org/10.22214/ijraset.2023.52867

[19] P. Bajpai and A. Lewis, "Secure development workflows in CI/CD pipelines," presented at the IEEE Secure Development Conference (SecDev). IEEE, pp. 65–66, 2022. https://doi.org/10.1109/secdev53368.2022.00024, 2022.

[20] C. Vassallo, S. Proksch, A. Jancso, H. C. Gall, and M. Di Penta, "Configuration smells in continuous delivery pipelines: A linter and a six-month study on GitLab," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, 2020. https://doi.org/10.1145/3368089.3409709*, 2020.

[21] A. Kaur and R. Nayyar, "A comparative study of static code analysis tools for vulnerability detection in c/c++ and java source code," *Procedia Computer Science*, vol. 171, pp. 2023-2029, 2020. https://doi.org/10.1016/j.procs.2020.04.217

[22] K. Kuszczyński and M. Walkowski, "Comparative analysis of open-source tools for conducting static code analysis," *Sensors*, vol. 23, no. 18, p. 7978, 2023. https://doi.org/10.3390/s23187978

[23] W. Charoenwet, P. Thongtanunam, V.-T. Pham, and C. Treude, "An empirical study of static analysis tools for secure code review," *arXiv*, 2024. https://doi.org/10.48550/ARXIV.2407.12241

[24] R. Paramitha and Y. D. W. Asnar, "Static code analysis tool for laravel framework based web application," presented at the International Conference on Data and Software Engineering (ICoDSE). IEEE, pp. 1–6,2021. https://doi.org/10.1109/icodse53690.2021.9648519, 2021.

[25] A. Efimov, S. Mishin, and E. Rogozin, "Forecasting the number of identified information security vulnerabilities based on the theory of" Gray Systems"," *Herald of Dagestan State Technical University Technical Sciences*, vol. 50, no. 3, pp. 72-82, 2023. https://doi.org/10.21822/2073-6185-2023-50-3-72-82

[26] S. SonarQube Customer Stories, "SonarQube customer stories, SkyScanner," Retrieved: https://www.sonarsource.com/resources/skyscanner/. [Accessed 2024.

[27] D. CheckMarx Case Study, "CheckMarx case study, DAZN," Retrieved: https://checkmarx.com/resources/case-studies/dazn-sca-english/. 2024.
[28] A. Mills, J. White, and P. Legg, ""OGMA: Visualisation for software container security analysis and automated remediation," presented at the IEEE International Conference on Cyber Security and Resilience (CSR). IEEE, 2022. https://doi.org/10.1109/csr54599.2022.9850335, 2022.
[29] S. H. Majumder, S. Jajodia, S. Majumdar, and M. S. Hossain, "Layered security analysis for container images: Expanding lightweight pre-deployment scanning," presented at the Annual International Conference on Privacy, Security and Trust (PST). IEEE, pp. 1–10, 2023. https://doi.org/10.1109/pst58708.2023.10320152, 2023.
[30] O. Javed and S. Toor, "An evaluation of container security vulnerability detection tools," presented at the International Conference on Cloud and Big Data Computing (ICCBDC). ACM, 2021. https://doi.org/10.1145/3481646.3481661, 2021.
[31] CyberArk, "Breaking down the codecov attack: Finding a malicious needle in a code haystack," Retrieved: https://www.cyberark.com/resources/blog/breaking-down-the-codecov-attack-finding-a-malicious-needle-in-a-code-haystack. 2024.
[32] A. Mycek and M. Lukaczyk, "Security of containerization platforms: threat modelling, vulnerability analysis, and risk mitigation," in *ECMS 2024 Proceedings edited by Daniel Grzonka, Natalia Rylko, Grazyna Suchacka, Vladimir Mityushev. ECMS, pp. 585–591, 2024. https://doi.org/10.7148/2024-0585*, 2024.
[33] V. Dakić, M. Kovač, and J. Slovinac, "Evolving high-performance computing data centers with kubernetes, performance analysis, and dynamic workload placement based on machine learning scheduling," *Electronics*, vol. 13, no. 13, p. 2651, 2024. https://doi.org/10.3390/electronics13132651
[34] Z. Moric, V. Dakic, and M. Kulic, "Implementing a security framework for container orchestration," presented at the IEEE 11th International Conference on Cyber Security and Cloud Computing (CSCloud). IEEE, pp. 200–206, 2024. https://doi.org/10.1109/cscloud62866.2024.00042, 2024.
[35] V. Dakić and A. Bubnjek, "Managing cloud-native applications using vSphere with Tanzu and Tanzu Kubernetes grid," *Edelweiss Applied Science and Technology*, vol. 8, no. 6, pp. 6557-6578, 2024. https://doi.org/10.55214/25768484.v8i6.3409
[36] K. Brady, S. Moon, T. Nguyen, and J. Coffman, "Docker container security in cloud computing," presented at the Annual Computing and Communication Workshop and Conference (CCWC). IEEE, 2020. https://doi.org/10.1109/ccwc47524.2020.9031195, 2020.
[37] K. German and O. Ponomareva, "An overview of container security in a kubernetes cluster," presented at the IEEE Ural-Siberian Conference on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT). IEEE, pp. 283–285, 2023. https://doi.org/10.1109/usbereit58508.2023.10158865, 2023.
[38] P. Mytilinakis, ""Attack methods and defenses on Kubernetes," Jun. 2020, University of Piraeus," Retrieved: https://doi.org/10.26267/UNIPI_DIONE/311. 2020.
[39] W. S. Shameem Ahamed, P. Zavarsky, and B. Swar, "Security audit of docker container images in cloud architecture," presented at the International Conference on Secure Cyber Computing and Communications (ICSCCC). IEEE, 2021. https://doi.org/10.1109/icsccc51823.2021.9478100, 2021.