Feature Flag Misconfiguration Vulnerabilities in Microservices Penetration Testing: Analysis, Impact, and Remediation

Ahmad H. Al-Omari
Cyber Security Department, Faculty of Science and Information Technology, Al-Zaytoonah University of Jordan, Jordan
a.alomari@zuj.edu.jo (corresponding author)

Ahmad Alshanty
Cyber Security Department, Faculty of Science and Information Technology, Al-Zaytoonah University of Jordan, Jordan
a.alshanty@zuj.edu

Received: 22 September 2025 | Revised: 20 October 2025, 30 October 2025, and 3 November 2025 | Accepted: 4 November 2025

 DOI: https://doi.org/10.48084/etasr.15014

ABSTRACT

This study examines the security implications of the adoption of Feature Flags (FFs) within microservice-based software development methodologies. FFs enable developers to activate or deactivate application functionality without redeploying code, thus accelerating service delivery and adoption. However, this approach introduces significant security risks, as misconfigured or poorly managed FFs can create novel vulnerabilities that attackers may exploit, such as bypassing critical security controls. To investigate these risks, a penetration testing analysis was conducted in a production-grade financial microservices environment, focusing on authentication mechanisms, fraud detection systems, and API logging. The assessment identified four high-severity vulnerabilities: FF misconfiguration, overly permissive privileges, insecure storage of API tokens, and persistent backdoors through shadow flags. Severity evaluations using the Common Vulnerability Scoring System (CVSS) revealed scores ranging from 8.8 to 9.3, indicating critical impact. These vulnerabilities were found to facilitate unauthorized financial transactions, data exposure, and regulatory noncompliance. In response to these findings, this study proposes a mitigation framework designed to systematically address FF vulnerabilities. This framework integrates a structured taxonomy, root cause analysis, and targeted remediation strategies, including enhanced role-based access controls, secure token management protocols, and automated auditing mechanisms.

Keywords-feature flags; microservices; penetration testing; vulnerability taxonomy; access control; API token management

I. INTRODUCTION

The architectural software design of microservices divides applications into discrete components that can be independently deployed, significantly improving development agility and systemic adaptability [1]. Microservice-based systems provide a flexible solution for running applications, where each service runs autonomously, so service failures do not affect other services [2]. Central to this model is the use of Feature Flags (FF), a practice that allows teams to toggle functionality for users instantly without the overhead and risk of redeployment [3]. By decoupling deployment from release, this strategy fosters iterative testing, safer release cycles, and the integration of in-progress features directly into production [4]. However, this accelerated development pace expands the potential attack surface of applications, which requires robust security measures [5]. In response to the expanding threat landscape [6], organizations are increasingly prioritizing security validation, with penetration testing emerging as a critical methodology. This involves simulating controlled attacks on a system to proactively identify vulnerabilities that could be exploited to breach integrity or authenticity [7]. This is instrumental both in remediating security flaws and in evaluating a system's resilience against unforeseen adversarial attacks [8].

In this context, this study elaborates on the stages, techniques, and tools of penetration testing, including the integration of web application firewalls, specifically within modern, agile development lifecycles. This approach mitigates risk by maintaining precise control over feature visibility and user access. This decoupling strategy supports controlled release management and experimental testing in live environments. This capability significantly speeds up the

process of software development, provides strong support for Continuous Integration and Delivery (CI/CD), and enables important practices such as split testing (A/B testing) and canary releases. Split testing compares two different versions of an application to evaluate their performance, while canary releases slowly introduce new features to a limited number of users before expanding to everyone. Although these abilities greatly improve service flexibility and encourage innovation, they also create serious potential security threats that older development methods cannot handle.

Secure authentication mechanisms, including complex text-based passwords and One-Time Passwords (OTP), constitute a foundational element of computerized system security [9]. However, the introduction of FFs presents a different class of risk, as their misconfiguration or mismanagement can create a significant attack surface within microservice architectures [10]. Unlike traditional vulnerabilities that arise from coding errors or infrastructure weaknesses, the perils associated with FFs typically originate from deficiencies in the management of their activation states and access control permissions. This mismanagement can inadvertently disable critical security controls, such as authentication procedures, fraud detection systems, and audit logging, or create pathways for their circumvention.

FFs have been widely studied for their operational benefits, but their security implications remain underexplored [11]. In [10], feature toggle usage patterns were analyzed, highlighting the risks associated with uncontrolled toggle proliferation, which can increase the attack surfaces. Similarly, in [12], it was demonstrated how FFs could be exploited to bypass authentication mechanisms if not properly managed. More recently, in [3], a framework for secure FF management was proposed, emphasizing access control and audit trails to prevent unauthorized toggling.

Microservice architectures introduce unique security challenges due to their distributed nature. In [13], a vulnerability assessment of microservice communication channels identified weaknesses in API security and token management. In [14], the focus was on privilege escalation risks in microservices, recommending Role-based Access Control (RBAC) improvements. This work extends these studies by specifically focusing on FF-related vulnerabilities within microservices.

In [15], it was stated that ensuring the security of microservice-based applications is still a challenge, especially since automatic detection of security problems in microservice-based applications is still an open area of research. This study introduced the term security smells to describe the security issues that microservice-based applications might suffer due to poor implementation. For example, an unauthenticated traffic smell can occur when a microservice accepts external or internal client requests without proper authentication. Another example is multiple user authentication, which happens if an intruder could be authenticated as an end user, since microservice applications are authenticating users through multiple different access points, which can be exploited by an intruder to authenticate as a legitimate user. In [16], a set of analytical techniques was proposed to automatically detect such smells by introducing KubeHound, a flexible prototype tool designed to identify these smells in microservice-based applications.

Penetration testing remains a fundamental method for proactive security evaluation [17]. In [18], automated penetration testing tools were specifically designed for microservices, facilitating dynamic discovery of vulnerabilities. In [19], the critical importance of integrating security testing into CI/CD pipelines was emphasized to identify misconfigurations at an early stage. This work builds on the methods established in these studies by incorporating dedicated test cases for FFs and real-world exploitation scenarios to evaluate their unique security implications.

This study investigates these security risks through rigorous penetration testing on live microservices responsible for authentication, OTP validation, fraud detection, and logging. The research identifies several high-impact vulnerabilities, including insecure storage of feature flag API tokens, excessive privileges granted to development and service accounts, and the persistence of unmonitored shadow flags that evade audit controls. In this way, organizations may unintentionally expose sensitive data and permit unauthorized actions, thus noticeably increasing the overall risk to their systems [20]. These vulnerabilities collectively threaten the confidentiality, integrity, and availability of microservices applications. The contributions of this study are as follows:

- A comprehensive taxonomy of FF vulnerabilities in microservices environments.
- A quantitative analysis of vulnerability impact and severity using the Common Vulnerability Scoring System (CVSS) framework.
- A root cause analysis that elucidates systemic weaknesses in FF governance.
- Practical remediation and recommendations to mitigate identified risks.
- A comparative evaluation against existing studies in FF security and microservices vulnerability management.

II. METHODOLOGY

The dataset used in this study was sourced from a live, production-grade microservices architecture that supports core financial services, including transaction processing, user authentication, fraud detection, and system logging. The dataset contains multiple components, such as configuration files, API logs, and access policies. The data that support the findings of this study are available from the corresponding author upon reasonable request. Together, all components provide a comprehensive overview of the system design and how it actually operates in a real financial setting. FFs were used in the system to control important security functions, such as OTP verification and automated fraud detection.

It is important to note that this dataset is highly confidential. Special access was granted to this study under strict agreements that protect the data, and all necessary ethical and corporate approvals were obtained beforehand. Due to the

sensitive and proprietary nature of this financial data, the raw dataset is not available publicly. The testing was conducted on a Kubernetes-orchestrated digital wallet platform comprising 47 microservices. The tested snapshot included security-critical services (auth, OTP, fraud) and a feature-flagging system with 312 flags, evaluated using an Open Feature-compatible SDK. The tool versions, including OWASP ZAP 2.14.0 and Jenkins 2.414.x LTS, were pinned for reproducibility.

A. Ethical Considerations and Data Protection

All activities were carried out with formal approvals and confidentiality agreements in a secure staging environment that mirrored production. Test accounts and bounded test transactions focused on control-plane effects (flag states, RBAC, CI/CD tokens) rather than customer data extraction. Each change was recorded, and the pre-test states were restored immediately after each experiment.

B. Tools and Limitations

Penetration testing was performed in a secure staging environment that exactly mirrored the live production system. This test environment used containerized microservices managed by Kubernetes, including a central console and API endpoints for managing FF. The penetration testing utilized the following tools:

- OWASP ZAP for automated vulnerability scanning.
- Postman for crafting and executing API requests.
- Jenkins for CI/CD pipeline analysis.
- The Burp suite for intercepting and manipulating HTTP traffic.
- Custom scripts for FF state manipulation and audit log analysis.

However, this research has the following limitations:

- The testing was performed on a single organizational environment, potentially affecting generalizability.
- Some proprietary configurations could not be disclosed due to confidentiality agreements.
- Dynamic FF behaviors depending on runtime conditions may not be fully captured.

C. External Validity and Applicability

Although this assessment was conducted in one organization, the approach is technology-agnostic and can be transferred directly to other microservice-based financial systems. The framework targets common touchpoints, flag governance and RBAC for write scopes, CI/CD token storage and rotation, and controlled rehearsal of flag-state changes in staging using standard toolchains (Burp/ZAP, Postman, Jenkins). Adoption mainly requires mapping these checks to a team's service inventory and flag catalog. The remediation measures proposed, namely, classification, environment segregation, RBAC narrowing, vaulting/rotation, and shadow-flag scanning, are likewise generic and pipeline-friendly.

III. FINDINGS AND RESULTS

The vulnerabilities identified through penetration tests of FF management in a microservices environment were analyzed. Vulnerabilities were classified by type, evaluated for criticality and risk, and assessed for impact and severity using established benchmarks. Finally, a root cause analysis was conducted to reveal the underlying factors that contribute to these security issues.

A. Vulnerability Taxonomy

An integrated methodological framework was employed for vulnerability assessment, combining a quantitative scoring model with a qualitative taxonomic classification. This dual approach was designed to produce both a systematic, metrics-based evaluation of severity and a functional categorization to inform remediation strategies. The methodological core resides in the application of CVSS v3.1, which establishes the quantitative foundation for all subsequent analysis [21].

The CVSS v3.1 framework generates objective, standardized severity scores based on an evaluation of vulnerability properties. These properties are formalized through exploitability metrics such as Attack Vector (AV), Attack Complexity (AC), Privileges Required (PR), User Interaction (UI), and impact metrics. The AC metric was assigned a value of either Low or High based on the CVSS specification. Weights of 0.77 for Low and 0.44 for High were algorithmically incorporated into the exploitability sub-score. This design ensures that vulnerabilities requiring complex, unpredictable exploitation conditions are automatically assigned a lower severity score, thus providing a data-driven basis for prioritization. For example, a high prevalence of vulnerabilities within the FF misconfiguration category directly indicates a systemic weakness in configuration management protocols, an insight that is supplemental to the individual CVSS scores. Table I presents a taxonomy of FF vulnerabilities used in this study.

TABLE I. VULNERABILITIES TAXONOMY

Vulnerability type	Description	Severity
FF misconfiguration	Bypass of security controls via toggle misuse	High
Overly permissive privileges	Unrestricted write access to production flags	High
Insecure storage of API tokens	Plaintext storage of tokens in CI/CD pipelines	High
Persistent backdoor via shadow flags	Undetected flags enabling long-term access	High

CVSS v3.1 scores (8.8–9.3) align with the business impact observed in payment workflows. The vector components AV, AC, PR, UI, and C/I/A were explicitly mapped to loss scenarios that matter to operators: unauthorized transactions from OTP-related control weakening, non-owner flag tampering through over-permissive roles, programming misuse of flags from exposed CI/CD tokens, and persistence through shadow flags that evade routine monitoring. The tight clustering indicates operationally similar risk across distinct exploit paths, which is consistent with threat models for crown-jewel payment flows and supports a clear remediation priority.

The two methodological components are intrinsically merged: the CVSS score provides the "why" for prioritization (i.e., which vulnerabilities pose the most immediate threat), while the taxonomy provides the "how" for mitigation (i.e., identifying common patterns to address vulnerabilities collectively). This integrated approach ensures that the classification is not only precise in its risk assessment but also operationalizable for strategic security response. Figure 1 presents the distribution of vulnerabilities by type, allowing straightforward comparison across categories and highlighting the most frequently observed vulnerability types.

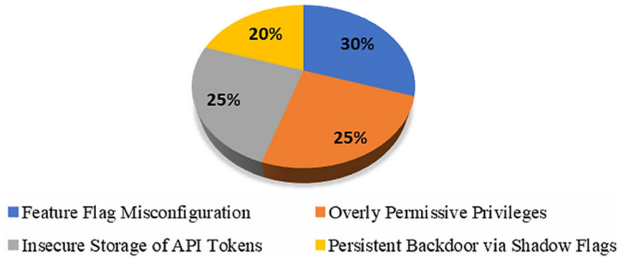

Fig. 1. Vulnerability distribution.

B. Critical Vulnerabilities

Vulnerabilities were classified and examined as critical for their potential impact on system security and operational integrity. Each vulnerability was evaluated with respect to its effect on confidentiality, integrity, and availability, with assessments supported by CVSS metrics. Table II provides a detailed view of critical vulnerabilities, linking each to its impact, CVSS score, and root cause. The CVSS scores, all above 8.8, indicate critical severity, reflecting the ease of exploitation and the profound consequences of these vulnerabilities.

TABLE II. VULNERABILITIES IMPACTS

Vulnerability type	Impact description	CVSS score	Root cause
FF misconfiguration	OTP bypass leading to unauthorized transactions	9.1	Lack of toggle criticality control
Overly permissive privileges	Unauthorized modification of security flags	9.3	Inadequate RBAC design
Insecure storage of API tokens	Exposure of API tokens enabling covert flag manipulation	9.0	Unencrypted token storage
Persistent backdoor via shadow flags	Persistent shadow flags evading audits and enabling backdoors	8.8	Absence of code review and scans

C. High-Risk Vulnerabilities

Vulnerabilities typically represent varying levels of risk to business operations and data security. Therefore, understanding their specific exploitation vectors and severity impacts is critical. As shown in Table III, the identified vulnerabilities range from high-risk threats to those with a lower severity level.

TABLE III. VULNERABILITIES' EXPLOITATION VECTORS AND SEVERITY IMPACT

Vulnerability type	Exploitation vector	Severity impact
FF misconfiguration	API PATCH requests disabling OTP and fraud checks	High risk of financial fraud and data breach
Overly permissive privileges	Use of "Project Editor" tokens to alter production flags	Enables unauthorized fund transfers and monitoring evasion
Insecure storage of API tokens	Extraction of tokens from Jenkins configs and environment	Full programmatic control over feature flags
Persistent backdoor via shadow flags	Creation of benign-named flags targeting attacker segments	Long-term persistence of unauthorized access

Figure 2 characterizes high-risk vulnerabilities by mapping their exploitation vectors to the severity of their potential impact on business operations and data security. The highest risk is unauthorized transfer, with a risk of 30%, followed by financial fraud and data breach with a rate of 25% for each, and at the lowest rates are the monitoring evasion and the long-term access with 10% each.

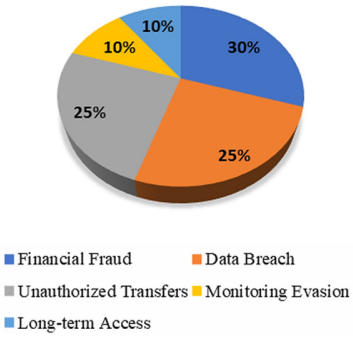

Fig. 2. Exploitation vectors.

D. Vulnerability Impact and Severity Analysis

A quantitative analysis of vulnerabilities was performed using CVSS v3.1. This framework provides a standardized method for assessing severity by evaluating exploitability, impact, and scope metrics. The resulting scores are instrumental in establishing an objective prioritization of remediation efforts [18]. Scores above 9.0 were observed to indicate critical severity, reflecting the potential for complete compromise of confidentiality, integrity, and availability. For example, PT-H02's score of 9.3 is driven by its exploitability, impact on financial assets, and lack of required privileges for exploitation.

E. Root Cause Analysis

Effective remediation of vulnerabilities necessitates a thorough understanding of their underlying root causes. The analysis identified several systemic issues that contribute to FF security weaknesses, including:

- Misclassification of FFs: Failure to categorize flags by criticality leads to indiscriminate toggling of sensitive controls.

- Inadequate Access Controls: Overly broad RBAC roles grant excessive privileges to non-administrative users.
- Poor Secrets Management: Storing API tokens in plaintext within CI/CD pipelines exposes them to theft.
- Lack of Audit and Review Processes: Shadow flags bypass code reviews and automated scans, enabling persistent backdoors.

Figure 3 shows a proportional distribution of the root causes that contribute to the observed vulnerabilities. Quantitative analysis reveals that inadequate access control is the most frequent root cause, followed, in descending order of prevalence, by misclassification, poor secrets management, and lack of audit and review processes.

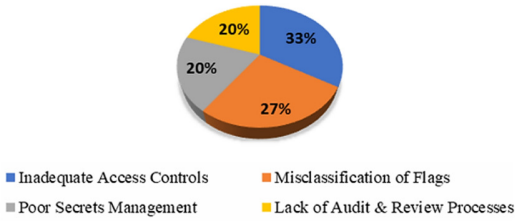

Fig. 3. Root cause frequency.

F. Comparison and Contribution

This work advances prior conceptual discussions by moving from theory to measured evidence. FF changes were tested in a production-grade wallet environment (Kubernetes, 47 microservices) using an OpenFeature-compatible SDK with over 300 flags, reproducing concrete exploit paths such as temporarily relaxing OTP or fraud checks via API-level flag manipulation. The impact was quantified using CVSS v3.1, with critical scores clustering between 8.8 and 9.3. All steps are reproducible under pinned tool versions. Together, exploit narratives, severity metrics, and artifacts convert the risks from conceptual to demonstrably measurable.

This work shows how these risks can be actively exploited in real systems through hands-on penetration testing, turning theoretical concerns into clear attack paths with measurable impact and demonstrating their real-world significance.

TABLE IV. COMPARISON WITH PREVIOUS STUDIES

Dimension	Previous studies	This study (findings)
Focus	Conceptual risks	Exploit-driven, empirical analysis
Vulnerability scope	Authentication bypass, API weaknesses	Misconfigurations, OTP bypass, token leakage, and shadow flags
Methodology	Mapping studies, detection frameworks	Penetration testing, exploit narratives
Severity	Qualitative only	CVSS v3.1 (8.8–9.3)
Root causes	General mentions	Ranked: Access control, misclassification, secrets, audit
Exploitation	Not demonstrated	Documented attack paths
Practicality	High-level recommendations	Actionable test cases, CI/CD integration
Contribution	Terminology and frameworks	Bridges theory and practice with measurable, exploitable scenarios

Previous works remained conceptual in their recommendations. This study delivers actionable test cases for CI/CD pipelines and practical remediation guidance, bridging the gap between research and security engineering practice. Table IV summarizes a comparison with previous studies. This work strengthens prior research by turning abstract risks into evidence-based, quantifiable, and actionable findings, providing both technical depth and practical security guidance.

IV. REMEDIATION AND RECOMMENDATIONS

Immediate remediation of the identified critical vulnerabilities necessitates an integrated strategy encompassing technical, procedural, and organizational dimensions. Table V shows vulnerabilities and recommended actions (remediations). The proposed framework addresses FF misconfigurations in microservices architectures. Core initiatives include establishing rigorous flag governance, implementing robust access control and secrets management, and adopting continuous auditing to maintain system integrity and prevent exploitation.

TABLE V. TABLE V. REMEDIATION MEASURES

Vulnerability type	Recommended Actions
FF misconfiguration	Classify feature flags by criticality; enforce least privilege; segregate production and staging.
Overly Permissive Privileges	Redesign RBAC to restrict write access to security flags; implement audit logging of changes.
Insecure Storage of API Tokens	Migrate tokens to secure vaults; generate environment-specific tokens; automate token rotation.
Persistent Backdoor via Shadow Flags	Enforce code review gates; implement automated scanning for unused or suspicious flags; require approvals.

This study operationalized "FF security as code" by integrating controls into CI/CD rather than relying on ad-hoc reviews. More specifically:

- Require criticality labels for flags and segregate production/stage environments so sensitive toggles receive elevated scrutiny.
- Narrow write access to high-impact flags via RBAC and maintain complete audit trails.
- Vault, scope, and automatically rotate flag tokens to eliminate plaintext exposure.
- Continuously surface unused or suspicious shadow flags and block releases when safety policies are weakened.
- Track simple KPIs, e.g., percentage of flags classified, number of privileged flag writers, token rotation age, and release blocks due to policy gates, to provide transparent, measurable assurance.

Together, these measures reduce the window for control-plane abuse in microservices and provide continuous assurance at release time.

The proposed remediation measures address core vulnerabilities through a multi-layered strategy that involves implementing an FF governance model with criticality-based classification, enforcing strict RBAC and environment

separation to prevent privilege escalation, and applying the principle of least privilege to restrict write access. This strategy further guarantees full traceability through comprehensive audit logging, secures API tokens via vault storage, scoped issuance, and automated rotation, and counters persistent backdoors with mandatory code reviews, automated flag scanning, and formal approval workflows for high-criticality flags, thus ensuring continuous oversight and preventing unauthorized modifications.

V. CONCLUSION AND FUTURE WORK

This study empirically identified critical security vulnerabilities arising from FF misconfigurations in microservices architectures, uncovering multiple high-severity weaknesses through controlled penetration testing. These vulnerabilities demonstrate a capability to facilitate authentication bypass, subvert fraud detection systems, and enable the extraction of sensitive data. By applying CVSS, this study provides a quantified assessment of vulnerability severity and impact, establishing an empirically grounded prioritization framework for remediation efforts.

Future work can focus on two primary directions: the development of automated tools for continuous security monitoring of FF configurations, and the direct integration of security policies into FF management platforms. Furthermore, extending this research to encompass a wider array of organizational environments will be essential for validating and enhancing the generalizability of the proposed findings and mitigation strategies.

ACKNOWLEDGMENT

The authors would like to thank the Deanship of Scientific Research and Innovation at Al-Zaytoonah University of Jordan (ZUJ) for funding this work through project no. 22/06/2025-2024

REFERENCES

[1] N. Nivedhaa, "Software architecture evolution: Patterns, trends, and best practices," *International Journal of Computer Sciences and Engineering (IJCSE)*, vol. 1, no. 2, pp. 1–14, Dec. 2024.

[2] F. H. Khoso, A. Lakhan, A. A. Arain, M. A. Soomro, S. Z. Nizamani, and K. Kanwar, "A Microservice-Based System for Industrial Internet of Things in Fog-Cloud Assisted Network," *Engineering, Technology & Applied Science Research*, vol. 11, no. 2, pp. 7029–7032, Apr. 2021, https://doi.org/10.48084/etasr.4077.

[3] D. Esther and E. Oliver, *Feature Flags and Dynamic Configuration in Microservices*. 2025.

[4] S. S. Ega and V. Motamarri, "Feature Flags and Configuration: Balancing Flexibility with Maintainability in Software Development," *Journal Of Engineering And Computer Sciences*, vol. 4, no. 8, pp. 751–760, Aug. 2025.

[5] R. Mahdavi-Hezaveh, J. Dremann, and L. Williams, "Software development with feature toggles: practices used by practitioners," *Empirical Software Engineering*, vol. 26, no. 1, Jan. 2021, Art. no. 1, https://doi.org/10.1007/s10664-020-09901-z.

[6] D. Bhatia, "A Comprehensive Review on the Cyber Security Methods in Indian Organisation," *International Journal of Advances in Soft Computing and its Applications*, vol. 14, no. 1, pp. 103–124, Apr. 2022, https://doi.org/10.15849/IJASCA.220328.08.

[7] R. Khalil, M. Wedyan, R. Saadeh, and R. Alturki, "A New Approach of Virtual Reality Systems Evaluation and Quality Standards," *International Journal of Advances in Soft Computing and its Applications*, vol. 16, no. 3, pp. 252–271, Nov. 2024, https://doi.org/10.15849/IJASCA.241130.14.

[8] R. Vallabhaneni and V. Veeramachaneni, "Understanding Penetration Testing for Evaluating Vulnerabilities and Enhancing Cyber Security," *Engineering and Technology Journal*, vol. 09, no. 10, Oct. 2024, https://doi.org/10.47191/etj/v9i10.12.

[9] S. Hamid, N. Z. Bawany, and S. Khan, "AcSIS: Authentication System Based on Image Splicing," *Engineering, Technology & Applied Science Research*, vol. 9, no. 5, pp. 4808–4812, Oct. 2019, https://doi.org/10.48084/etasr.3060.

[10] M. T. Rahman, L. P. Querel, P. C. Rigby, and B. Adams, "Feature toggles: practitioner practices and a case study," in *Proceedings of the 13th International Conference on Mining Software Repositories*, Austin, TX, USA, Feb. 2016, pp. 201–211, https://doi.org/10.1145/2901739.2901745.

[11] N. Dragoni *et al.*, "Microservices: Yesterday, Today, and Tomorrow," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds. Springer International Publishing, 2017, pp. 195–216.

[12] N. Alshuqayran, N. Ali, and R. Evans, "A Systematic Mapping Study in Microservice Architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, Macau, China August 2016, pp. 44–51, https://doi.org/10.1109/SOCA.2016.15.

[13] R. K. Jayalath, H. Ahmad, D. Goel, M. S. Syed, and F. Ullah, "Microservice Vulnerability Analysis: A Literature Review With Empirical Insights," *IEEE Access*, vol. 12, pp. 155168–155204, 2024, https://doi.org/10.1109/ACCESS.2024.3481374.

[14] A. R. Sinha, "Unified System Design: A Comprehensive Study on Scalability, Access Control, and Communication Protocols," *IJSAT - International Journal on Science and Technology*, vol. 15, no. 2, May 2024, https://doi.org/10.71097/IJSAT.v15.i2.2845.

[15] F. Ponce, J. Soldani, H. Astudillo, and A. Brogi, "Smells and refactorings for microservices security: A multivocal literature review," *Journal of Systems and Software*, vol. 192, Oct. 2022, Art. no. 111393, https://doi.org/10.1016/j.jss.2022.111393.

[16] G. Dell'Immagine, J. Soldani, and A. Brogi, "KubeHound: Detecting Microservices' Security Smells in Kubernetes Deployments," *Future Internet*, vol. 15, no. 7, July 2023, Art. no. 228, https://doi.org/10.3390/fi15070228.

[17] J. Edwards, "Vulnerability Assessment and Penetration Testing," in *Mastering Cybersecurity: Strategies, Technologies, and Best Practices*, J. Edwards, Ed. Berkeley, CA, USA: Apress, 2024, pp. 371–412.

[18] B. Ünver and R. Britto, "Automatic Detection of Security Deficiencies and Refactoring Advises for Microservices," in *2023 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, Melbourne, Australia, May 2023, pp. 25–34, https://doi.org/10.1109/ICSSP59042.2023.00013.

[19] M. Mangla, "Securing CI/CD Pipeline: Automating the detection of misconfigurations and integrating security tools," M.S. thesis, National College of Ireland, 2023.

[20] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, Feb. 2018, https://doi.org/10.1109/MS.2018.2141039.

[21] "CVSS v3.1 Specification Document." https://www.first.org/cvss/v3-1/specification-document.