Journal of Computational Analysis and Applications VOL. 34, NO. 12, 2025
10.48047/jocaaa.2025.34.12.82

Bridging the Gap Between Developers and Security Engineers in Automation Pipelines

Sameer Lakade
Independent Researcher, USA

1. Abstract

Modern software development has transformed through automation pipelines, enabling continuous integration and delivery, accelerating deployment frequencies to unprecedented levels. However, this velocity has exposed a critical challenge: traditional security review processes create bottlenecks that either delay releases or compromise security assurance. This article examines the fundamental misalignment between development and security teams, rooted in divergent incentives, fragmented toolchains, and separate communication channels that create organizational silos, undermining both productivity and security posture. To propose a comprehensive socio-technical approach to bridge this divide through multiple integrated strategies. First, security-as-code embeds automated security validations directly into development pipelines, utilizing static analysis, dynamic testing, dependency scanning, secret detection, and policy-as-code enforcement to catch vulnerabilities at the moment of introduction rather than weeks later. Second, unified observability platforms integrate security metrics into operational dashboards, creating shared visibility that enables both teams to understand system health holistically and develop mutual empathy for each other's challenges and constraints. Third, organizational transformation strategies, including shift-left security practices, security champion programs, shared performance metrics, and dedicated communication channels, align incentives and foster a collaborative culture. Finally, advanced automation infrastructure leveraging artificial intelligence and machine learning provides intelligent vulnerability classification, context-aware remediation guidance, and natural language explanations that make security findings actionable for developers while reducing false positive fatigue. This integrated approach demonstrates that when security evolves from post-development inspection to collaborative participation throughout the development lifecycle, organizations can achieve both rapid deployment velocity and robust security assurance, establishing the foundation for sustainable innovation in an era defined by automated software delivery.

Keywords: DevSecOps, Security-As-Code, Automation Pipelines, Shift-Left Security, Collaborative Visibility

2. Introduction

The radical change in modern software development has been brought about by automation pipelines, allowing code-based continuous integration, continuous delivery, and code-managed infrastructure. This has seen the software delivery speed up to levels never before witnessed, with organizations now able to roll out changes several times a day, unlike once a month or quarterly. The research on the scaling of agile transformation and DevOps maturity in large organizations states that high-performing teams deploy much faster than traditional development practices, and mature DevOps practices allow organizations to cut the lead times by weeks or months to hours or days without compromising the quality and reliability standards [1]. This rapid pace of development has, however, revealed one very serious issue: the old method of security review cannot keep up with the current development pace, and there is what industry participants

10.48047/jocaaa.2025.34.12.82

refer to as the security-speed paradox of which the same process of automation that makes it a feasible part of the development process can also make it a vulnerability at an unprecedented rate unless security concerns are directly built into the development cycle.

The inherent issue is that in the model of handoff, developers develop and make code changes at a very fast pace, and security teams examine the changes individually and usually days or weeks later. A study conducted to identify the effects of data breaches indicates that the financial and operating expenses of a security incident increase drastically when the vulnerabilities are identified in the later stages of the development lifecycle or even worse, once the vulnerability has been deployed to the production environments. This forms a bottleneck that slows down the releases either by making long security approval cycles, or results in hard tradeoffs in organizations between the speed of deployment and the assurance of security. The underlying cause is a poor fit of incentives; developers are being rewarded for innovation, and performance measures are consequently based on deployment frequency, feature delivery speed, and time to market; security professionals are being rewarded for risk minimization, and performance metrics are usually based on vulnerability reduction rate, compliance, and incident prevention. Research on the maturity of DevOps has shown that companies that have a dysfunctional relationship with this misalignment have much longer deployment times, greater rates of changes failing, and longer mean time to recovery in case of an incident, indicating that the cultural gap between security and development has a direct relationship with the efficiency of operations and security performance [1]. Such conflicting goals, as well as the division of tools and disjointed communication channels, form organizational silos that not only hinder productivity but also weaken the security posture. Developers usually use integrated development environments, version control systems, build automation tools, and deployment orchestration platforms, and security teams use independent vulnerability assessment tools, security information and event management systems, compliance monitoring platforms, and threat intelligence feeds. The result of such fragmentation of tools is that security results are usually presented in formats, dashboards, and reporting designs that are not familiar to a developer and which require considerable context switching, and manually aligning information across different systems, and may take a long time to translate security requirements to development tasks. According to the studies about the prices of data breaches, enterprises incur significant financial losses due to security breaches, where the expenses include not only the immediate remedies and response but also the reputational damage in the long term, regulatory fines, and business losses [2]. Moreover, the security requirements recorded in lengthy policy books, regulatory compliance schemes, or in architectural review board rulings are never connected with the executable infrastructure-as-code and application code that programmers write day in, day out, so they are misinterpreted, and security standards are not consistently applied, or a gap between the planned security controls and the implemented configurations.

The impact is often a delayed release of products that miss important market releases, frustrated coders who perceive security as a bottleneck to sustainable innovation, and, in some cases, the workaround that entirely violates security. According to organizations, developers occasionally build shadow automation pipelines, deploy via unapproved paths, or bypass obligatory security gates to uphold velocity guarantees, and unwillingly add unknown and unmonitored risks into production systems.

3. Integrating Security into Development Workflows

The answer is to put security in what it is meant to be, not as a reacting checkpoint, but as an active participant in the development pipeline. This starts with security-as-code, wherein security verifications are now automated processes and no longer manual verification gates that introduce delays and bottlenecks in the release process. Experiments conducted on automated security testing in DevOps setups based on

10.48047/jocaaa.2025.34.12.82

artificial intelligence and machine learning indicate that entities implementing automated security scanning in their continuous integration and continuous delivery pipelines have the benefit of seeing a major increase in the rate at which vulnerabilities are identified and the rate at which software vulnerabilities are discovered, and that AI-based tools can help to detect more sophisticated security vulnerabilities that otherwise would not be noted during traditional static analysis and reduce false positive rates that tend to plague manual security scanning processes [3]. Before the code is integrated into the main branch, static analysis can be used to detect vulnerabilities in the code. Source code can be scanned to identify typical security vulnerabilities, including SQL injection vulnerabilities, cross-site scripting, breaches of buffer integrity, cryptographic or weak cryptographic implementations, and the use of untested input validation. These tools can analyze the code patterns, data flow, and control flow comprehensively to detect possible areas of security vulnerability without execution of the program, allowing developers to get prompt and precise feedback on the active coding stage before vulnerabilities are sealed in the codebase.

Dynamic testing is used to test running applications to detect security bugs by testing deployed applications against attack scenarios simulating real-life situations and detecting dynamically vulnerable programs that are not detectable by static analysis because they require application execution contexts. This consists of authentication bypass, authorization, and privilege escalation bugs, session management bugs, API security bugs, and injection bugs, which can only be demonstrated once the application receives real user requests and communicates with the backend systems and databases. Dependency scanners determine vulnerable third-party libraries by systematically matching project dependencies with large databases of known vulnerabilities, and modern applications are often full of hundreds or thousands of third-party components, frameworks, and libraries that they need to constantly search to identify any new security vulnerabilities. By incorporating machine learning algorithms into automated security testing processes, these systems can enhance their capabilities to learn and react to changes in the threat environment as well as prioritize security results in accordance with the actual risk environment and not according to generic vulnerability severity indicators, which greatly enhances the efficiency of security validation procedures [3].

Secret detection does not store credentials in repositories by automatically scanning code changes in patterns that match API keys, passwords, cryptographic private key, database connection string, authentication tokens, and other sensitive configuration data. These advanced tools utilize patternmatching algorithms, entropy analysis to identify high-randomness strings that are indicative of secrets, and machine learning models that identify sensitive data that developers may leave in source code, configuration files, or documentation. Studies into the occurrence and security impact of credential leakage in repositories indicate worrying data concerning the detection of thousands of leaked secrets in the analyzed repositories, encompassing database credentials, API tokens, private encryption keys, and cloud service access credentials that may be used to gain unauthorized access to important systems and data [4]. To ensure image security prior to deployment, container scanning is done to verify the image security using multi-layer analysis of container images, base operating system images, application dependencies, system libraries, installed packages, and configuration files to guarantee that containers are secure and meet organizational security aspects and compliance requirements before being released into production environments.

The policy-as-code, correspondingly, is also vital and converts the security and compliance requirements into machine-readable languages that can be automatically enforced as part of the building and deployment process, without manual code and subjective interpretation it subjectively. Organizations can specify the exact rules in domain-specific policy languages that automatically verify infrastructure configuration, access control policy, network security settings, encryption standards, logging policies, and deployment parameters, instead of keeping long policy documents, compliance checklists, and security standards

10.48047/jocaaa.2025.34.12.82

documentation, which need human interpretation and manual verification by security teams. The pipeline is self-enforcing as it automatically identifies and stops security problems at the very moment they are introduced in development, instead of identifying the problems weeks or months later through security audits, penetration testing, or after being deployed to production environments, where the cost of remediation and business impact are exponentially increased.

Validation Aspect	Manual Documentation Approach	Policy-as-Code Approach	Advantage Factor
Consistency across environments	Variable (human interpretation)	Identical (automated enforcement)	100% consistency
Validation speed	Days to weeks	Seconds to minutes	1000x+ faster
Interpretation ambiguity	High (subjective reading)	Zero (executable specifications)	Complete elimination
Coverage completeness	Partial (sampling)	Complete (every deployment)	Total coverage
Feedback timing	Post-deployment or audit	Real-time during build	Immediate
Human resource requirement	High (manual review)	Low (automated execution)	80-90% reduction
Scalability	Limited by team size	Unlimited (automated)	Infinite scaling

Table 1: Policy-as-Code vs. Traditional Manual Security Validation: Efficiency, Consistency, and Scalability Comparison [3, 4]

4. Building Collaborative Visibility

Effective collaboration requires shared understanding, which demands unified observability across development and security domains. Traditional approaches keep these teams working with separate dashboards, monitoring different metrics, and lacking visibility into each other's concerns, creating information silos that prevent a holistic understanding of system health and security posture. Research on enhancing collaboration between development and operations teams in DevOps demonstrates that organizations implementing unified monitoring and visibility platforms experience significant improvements in cross-functional collaboration, with integrated observability tools serving as critical enablers for breaking down traditional silos and fostering shared responsibility for both system reliability and security outcomes [5]. Modern approaches integrate security metrics directly into operational dashboards, showing vulnerability counts, security scan results, and compliance status alongside deployment frequency, build success rates, application performance indicators, infrastructure health metrics, and user experience measurements. This shared visibility creates common ground for discussion and enables both teams to understand how their work affects broader organizational objectives, fostering a culture of collective responsibility where security and operational excellence are viewed as complementary rather than competing priorities.

Security observability extends beyond simple vulnerability counts to include comprehensive policy compliance status tracking, real-time threat detection alerts, behavioral anomaly identification, and access pattern analysis that reveals both legitimate usage patterns and potential security incidents. Studies

10.48047/jocaaa.2025.34.12.82

examining collaboration patterns in DevOps environments indicate that when security metrics are integrated into development workflows and presented in developer-friendly formats within the tools developers already use daily, teams demonstrate higher engagement with security issues, faster vulnerability remediation cycles, and more proactive security practices embedded directly into their routine work activities rather than treating security as a separate, disconnected concern [5]. When developers can see security metrics in their existing tools—including integrated development environments, continuous integration dashboards, pull request interfaces, and deployment monitoring systems—and security engineers can view deployment pipelines, code change histories, feature release schedules, and operational metrics, both teams develop empathy for each other's challenges, constraints, and priorities. Developers gain appreciation for the complexity of maintaining security across rapidly evolving systems with expanding attack surfaces and continuously discovered vulnerabilities, while security engineers understand the business pressures, technical constraints, architectural limitations, and operational realities that drive development decisions and deployment schedules.

This transparency reduces friction and enables faster resolution when issues arise, as both teams work from the same authoritative information sources, shared context, and unified data models rather than debating whose data is correct, which metrics matter most, or how to interpret conflicting signals from disparate monitoring systems. Research on software supply chain security reveals the complexity of modern dependency management, with analysis of package ecosystems showing that vulnerabilities in widely-used dependencies can affect thousands of downstream projects, making shared visibility into dependency health and security status critical for both development and security teams [6]. Furthermore, unified observability enables proactive identification of patterns and trends that might be invisible when data remains siloed, such as correlations between deployment frequency and vulnerability introduction rates, relationships between code complexity metrics and security flaw density, or connections between infrastructure configuration changes and security policy violations. The study of weak links in software supply chains demonstrates how transitive dependencies and maintainer relationships create complex security risk propagation paths that require comprehensive visibility to understand and manage effectively [6]. This holistic view supports data-driven decision making about security investments, risk acceptance criteria, and the appropriate balance between development velocity and security assurance, moving conversations from subjective opinions and departmental politics to objective analysis of measurable outcomes and organizational impact.

Metric Category	Traditional Siloed Approach	Unified Observability Approach	Improvement Area
Cross-functional collaboration effectiveness	Separate dashboards	Integrated observability tools	Breaking down silos
Visibility scope	Single domain metrics	Multi-domain metrics (security + operations)	Holistic system understanding
Security engagement level	Low (separate concern)	High (embedded in daily workflow)	Developer security engagement
Vulnerability remediation speed	Slow (fragmented data)	Fast (shared context)	Incident resolution time

10.48047/jocaaa.2025.34.12.82

Decision-making basis	Subjective opinions	Data-driven analysis	Objective organizational impact
Information source consistency	Conflicting signals	Unified data models	Reduced friction
Empathy between teams	Limited understanding	Mutual appreciation of constraints	Cross-team collaboration

Table 2: Comparative Analysis of Traditional vs. Unified Observability Approaches in DevSecOps Environments [5, 6]

5. Organizational Transformation Strategies

Technical integration cannot be achieved without the related organizational adjustments, which will cover the cultural, structural, and incentive mismatch between the development and security organizations. The shift-left security model advances validation to an earlier stage in the development lifecycle, based on pre-commit hooks and local scanning tools, integrated development environment extensions that identify security bugs before the code even reaches the shared repository or continuous integration pipeline. Empirical literature on DevOps best practices highlights that moving security left in the software development lifecycle is one of the core values of organizations that aim to balance speed with assurance of security, and early detection and remediation of security vulnerabilities is one of the critical success factors to successful implementations of mature DevOps practices [7]. This helps to eliminate expensive rework cycles, avoid security debt, and educates developers about good security coding practices by giving them simple feedback on the moment of creating the code, not delayed feedback that will arrive days or weeks later and require the developer to context-switch to get the problem fixed and to continue working. The immediate feedback loop establishes a strong learning process in which security developers learn to write secure code by repetition and reinforcement, slowly becoming security experts as an inseparable part of their development capabilities and not going through security as a checklist that a separate team enforces. Security champion programs formalize any security knowledge into a development team through the appointment of individual developers as security champions, security liaisons and subject matter experts who intersect between central security teams and the distributed development teams. These champions are also trained in specific code practices, threat modeling, and testing approaches, and the company's security policies and compliance needs, which allow them to be the point of security contact in their respective teams, review security-sensitive code updates, and give advice on security architecture choices. Studies to determine how developers find and distribute security and privacy advice on online platforms show that developers often face security-related issues and problems in their daily tasks, and research indicates that large numbers of security and privacy-related questions exist on developers' forums, and knowledge of security expertise is obviously required in the development domain [8]. This model not only turns security into an external liability imposed by a different department but also an internal resource that is part of development teams, developing distributed security expertise that scales better than centralized review models and creating a culture where security turns into the responsibility of everyone in the organization as opposed to the special prerogative of specialized security personnel. The champion model also tackles the gap of accessibility in studies of the behavior of seeking information by developers, where developers seek immediate, localized advice from trusted peers rather than consulting an external, remote security team or searching voluminous policy documentation [8].

10.48047/jocaaa.2025.34.12.82

Shared goals and service level agreements can coordinate the incentives by establishing measures that are important to both the development and security teams. Instead of conflicting priorities, you have collaborative ones to be reached together. Vulnerability remediation time between discovery and fix deployment, success percentages in security gates that reflect not only effectiveness in security but also experience in the domain of development and compliance success metrics that indicate compliance with regulations without slowing down the velocity of business creates accountability on both ends of the traditional development-security divide. DevOps best practice guides incorporate the idea that effective DevSecOps transformation does not only involve technological integration of tools, but also ultimate alterations in organizational framework, performance measurement frameworks and patterns of team association with shared metrics emerging as a key process through which previously disjointed goals can be brought into focus [7]. The shared post-incident retrospective views of security incidents contribute to organizational learning over blame allocation and build psychologically safe settings in which teams can frankly discuss their mistakes, can find systemic problems and not individual failures, and can take preventive actions based on root causes. Specialized communication tools like shared chat rooms, collaborative documentation tools, and informal virtual meeting rooms allow rapid sharing of information, quick resolution of security issues, and continuous communication without the baggage of formal meetings, approval processes, or time-consuming email traffic that not only slows down the development but also slows down security work.

Implementation Stage	Security Validation Point	Timing of Feedback	Cost of Remediation	Learning Effectiveness	Security Debt Impact
Pre-commit hooks	Before the code reaches the repository	Immediate (seconds)	Minimal	Very High	Prevention
Local scanning tools	During code creation	Real-time	Very Low	High	Prevention
IDE plugins	At the moment of coding	Instant	Lowest	Very High	Prevention
CI pipeline scanning	After code commit	Minutes to hours	Low to Medium	Medium	Early Detection
Testing phase review	During the QA cycle	Hours to days	Medium	Low	Accumulation
Production discovery	Post-deployment	Days to weeks	Very High	Very Low	High Accumulation

Table 3: Shift-Left Security Implementation and Impact Analysis [7, 8]

6. Advanced Automation Infrastructure

The infrastructure has to have the ability to work together by integrating the toolchains into the underlying infrastructure that links the development platforms to security systems and flows information between tools and workflows that were previously isolated. These centralized logging and configuration management

10.48047/jocaaa.2025.34.12.82

guarantee that both teams will be able to see the information that they need without traversing several unconnected systems, and remove the context switching overhead and information discontinuities that hinder productive teamwork. Studies focused on ongoing integration and ongoing delivery platform development indicate that the companies deploying integrated automation infrastructures achieve a dramatic increase in software quality, deployment rates, or effectiveness of teamwork, and the integrated platforms can support automated build systems, end-to-end testing pipelines, and automated deployment processes that lower the level of human-intervention and minimize the risk of a human error and shorten the software delivery cycles [9]. Role-based access control and zerotrust design are both non-obtrusive in creating operational friction, and both apply the principle of leastprivilege access controls that only provide users and systems with the minimum necessary permissions to complete their particular tasks, with a full audit trail of all access and activities. Ephemeral credentials, automatic credential rotation systems, and just-in-time provisioning of access keep both the security level high and the productivity of developers high by removing long-lived, static credentials, which can be used to signify a persistent security threat, and lower the delays and bottlenecks inherent to other access control methods, which require long-lived, non-ephemeral credentials.

Machine learning and artificial intelligence can facilitate collaboration by offering smarter vulnerability classification as a prioritization of security results on the basis of realistic exploitability, business context, and impact, instead of generic severity scores, which do not consider organization-specific risk elements. The context-sensitive remediation recommendations analyze the surrounding code, application architecture, and development patterns to offer explicit, practical recommendations, which can be implemented instantly by developers instead of general security recommendations, which demand considerable interpretation and translation of general recommendations. Malicious code detection study based on machine learning illustrates how effective feature extraction and classification methods are at detecting security risks and that the machine learning models are capable of analyzing the characteristics of the malicious code, its behavioural patterns, and structural properties to differentiate between a benign and malicious code with a high accuracy rate [10]. Natural language explanations that make security findings usable by developers are transformations of the technical vulnerability report into a clear, developer-friendly description of what the security issue is, why it is important, how it can be used, what specific code changes would fix it, and how to avoid the occurrence of similar problems in future development work. These intelligent systems study vulnerability patterns over the entire codebase, detect common security anti-patterns that represent either a lack of knowledge or weaknesses in processes, and feed developers with specific educational content, which assists them in achieving security expertise in the long run.

These systems minimize false positive fatigue and also make certain critical issues be addressed correctly with the use of intelligent filtering, prioritization, and presentation mechanisms that help differentiate between theoretically vulnerable vulnerabilities with little to no real-world risk and those vulnerabilities that are actively exploited and that require immediate fixes. The feature extraction methods utilized in security analysis allow automated systems to detect slight evidence of malicious or vulnerable code patterns through the analysis of a set of such code features together, such as syntactic structures, semantic properties, and behavioral features, which, when combined, will evidence security implications that would not have been detected through shallow analysis. Moreover, continuous integration and delivery platforms offer the automation framework needed to integrate security scanning, testing, and validation in the entire software development cycle, and research has shown that educational deployments of these platforms have been able to achieve this by showing how automation can revolutionize software engineering practice through the direct integration of quality and security checks into the development process [9]. With more sophisticated

10.48047/jocaaa.2025.34.12.82

systems, subtle patterns and anomalies may be noticed (which would not be any easier to notice with human inspection), such as unusual code patterns that are not part of accepted secure practices of coding, dependency combinations, which form unforeseen security implications, or configuration drift, which distracts security posture over time.

Analysis Dimension	Traditional Rule-Based	AI/ML Feature Extraction	Improvement Factor	Detection Capability
Syntactic structure analysis	Pattern matching only	Multi-dimensional feature analysis	Comprehensive	Enhanced
Semantic property evaluation	Limited context	Deep contextual understanding	Significant	Superior
Behavioral attribute detection	Surface-level only	Complex behavior modeling	High	Advanced
Code characteristic examination	Single attributes	Multiple simultaneous characteristics	Very High	Holistic
Subtle pattern identification	Obvious patterns only	Hidden pattern discovery	Breakthrough	Anomaly detection
Configuration drift detection	Manual inspection	Automated continuous monitoring	Proactive	Real-time
Dependency risk assessment	Known CVE matching	Combination risk analysis	Predictive	Preventive

Table 4: Comparative Analysis of Traditional Rule-Based vs. AI/ML-Enhanced Security Code Analysis Capabilities [9, 10]

7. Conclusion

There is no other way to bridge the gap between the developers and security engineers than to implement a fundamental change that will focus on both the technical and cultural aspects of contemporary software delivery. The historic handoff system, where security reviews are done independently of the development process, poses bottlenecks that cannot be overcome in a system where there is a need to have multiple deployments per day. This article has shown that the answer is in integrating security into development processes by adopting automated security-as-code practices, creating cohesive observability platforms that offer similar visibility to both domains, deploying organizational initiatives like shift-left security and security champion programs that distribute knowledge and incentives, and using advanced automation infrastructure that is augmented by artificial intelligence to provide smart, actionable security advice. These are combined strategies that turn security into an outside limitation and internal capability, and a culture of coexisting and not competing between high deployment speed and high-security assurance is achieved. The facts that have been provided in the course of the current paper suggest that in the case when the organizations adopt these all-inclusive approaches, the vulnerability remediating process may go on much faster, the security debt may be lower, the cross-functional collaboration may be improved, and, finally, the balance between speed and risk-management that is necessary to ensure the competitive advantage may be

10.48047/jocaaa.2025.34.12.82

achieved. Now that software systems are progressively becoming more complex, and deployment cycles continue to increase, the concepts and recommendations identified above form a roadmap through which organizations, wishing to create automation pipelines, are not only fast and reliable but fundamentally and sustainably secure. The result is more than process improvement as it creates the basis of digital trust and organizational resilience in a time where the speed of software delivery and security excellence need to proceed as complementary requirements and not competitors.

References

[1] Utham Kumar Anugula Sethupathy, "Scaling Agile Transformations: A Roadmap for DevOps Maturity in Large Organizations," December 2017. [Online]. Available: https://www.researchgate.net/publication/394965639_SCALING_AGILE_TRANSFORMATIONS_A_ROADMAP_FOR_DEVOPS_MATURITY_IN_LARGE_ORGANIZATIONS

[2] Adil Hussain et al., COVID-19 and diabetes: Knowledge in progress. *Diabetes & Metabolic Syndrome: Clinical Research & Reviews*., 13 May 2020. https://pmc.ncbi.nlm.nih.gov/articles/PMC7349636/

[3] Bipin Gajbhiye, "Automated Security Testing in DevOps Environments Using AI and ML," June 2024. [Online]. Available: https://www.researchgate.net/publication/383522032_Automated_Security_Testing_in_DevOps_Environments_Using_AI_and_ML

[4] Alexander Krause et al., "Studying Prevention and Remediation Strategies Against Secret Leakage in Source Code Repositories"., 14 November 2022. https://arxiv.org/abs/2211.06213

[5] Vidyasagar Vangala, "Enhancing Collaboration Between Development and Operations Teams in DevOps," November 2024. [Online]. Available: https://www.researchgate.net/publication/388420990_Enhancing_Collaboration_Between_Development_and_Operations_Teams_in_DevOps

[6] Nusrat Zahan et al., What are Weak Links in the npm Supply Chain?**.,** 14 Feb 2022, **https://arxiv.org/abs/2112.10165**

[7] Justin Ogala, "A Complete Guide to DevOps Best Practices," February 2022. [Online]. Available: https://www.researchgate.net/publication/360066207_A_Complete_Guide_to_DevOps_Best_Practices [8] Mohammad Tahaei et al., "Understanding Privacy-Related Advice on Stack Overflow," Apr. April 2022. [Online]. Available: https://www.researchgate.net/publication/359034922_Understanding_Privacy-Related_Advice_on_Stack_Overflow

[9] Sendy Ferdian Sujadi et al., "Continuous Integration and Continuous Delivery Platform Development of Software Engineering and Software Project Management in Higher Education," April 2021. [Online].
Available:
https://www.researchgate.net/publication/368075906_Continuous_Integration_and_Continuous_Delivery_Platform_Development_of_Software_Engineering_and_Software_Project_Management_in_Higher_Education

[10] Liuan Zheng et al., Detection approach of malicious JavaScript code based on deep learning. 2023. *IEEE*. https://ieeexplore.ieee.org/document/10165039