Bridging Security and Agility: A Comprehensive Approach to Integrating Security Practices in Agile Development through DAST, LLMs, & Automation

Arpit U. Thool

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Application

Chris Brown, Chair
Bob Edmison
Muhammad Ali Gulzar
Na Meng
Justin Smith

January 15, 2026
Blacksburg, Virginia

Keywords: Software Engineering, Agile, Security Practices

Bridging Security and Agility: A Comprehensive Approach to Integrating Security Practices in Agile Development through DAST, LLMs, & Automation

Arpit U. Thool

(ABSTRACT)

Effectively integrating security practices within Agile software development is essential as software systems become complex and critical. While Agile methodologies are widely adopted for their responsiveness and efficiency, many security practices remain documentation-heavy and process-driven, creating friction with Agile's emphasis on frequent delivery, individuals, and interactions. This Ph.D. dissertation investigates the integration of security practices—particularly Dynamic Application Security Testing (DAST), used to identify critical real-time vulnerabilities in web applications—into Agile workflows, examining its perceived impact on development teams and processes. We first surveyed Agile practitioners to understand their perspectives on security integration, revealing both benefits and challenges in implementation. We then explored how Large Language Models (LLMs) could improve the comprehension of security testing outputs, demonstrating that LLM generated summaries enhance the accessibility and understanding of security alerts. Subsequently, an in-depth real-world case study of a Kanban-based Agile team integrating DAST into its Continuous Integration Continuous Development (CI/CD) pipelines uncovered practical obstacles—such as report complexity and workflow interruptions, alongside conditions that supported successful adoption, including increased automation and dedicated engineering support. Finally, the insights from these studies informed the development of *SafeAIMerge*, a CI/CD-based tool that integrates

DAST scanning with LLM-generated summaries to deliver actionable, developer-friendly security feedback within pull requests (PRs). Practitioner evaluations indicate that the tool reduces cognitive and emotional workload during vulnerability remediation, enhances security report understanding, and supports software developers in more efficient resolution of security issues. Together, these studies form a cohesive body of evidence demonstrating how security practices such as DAST, when supported by automated workflows, LLMs, and guided by practitioner-centered design, can be effectively embedded into Agile development.

Bridging Security and Agility: A Comprehensive Approach to Integrating Security Practices in Agile Development through DAST, LLMs, & Automation

Arpit U. Thool

(GENERAL AUDIENCE ABSTRACT)

Modern software is deeply embedded in everyday life, making security a critical concern. At the same time, many software teams rely on Agile development methods that emphasize speed and frequent updates. Traditional security practices are often difficult to fit into these fast-paced workflows, creating challenges for teams trying to remain both secure and efficient. This PhD dissertation focuses on how security practices can be effectively incorporated into Agile software development. We first examined how software developers perceive security, identifying common benefits as well as obstacles that make security practices difficult to adopt in Agile. Next, we investigated how LLM tools can help developers better understand security testing results, showing that LLM-generated explanations make security issues clearer and easier to address. We then studied a real-world Agile team integrating automated security testing—specifically Dynamic Application Security Testing (DAST), which checks running web software for security weaknesses—into their development process, uncovering both practical challenges and factors that supported successful adoption. Based on these findings, we developed *SafeAIMerge*, a DAST-based tool that provides clear and actionable security feedback directly within developers' existing workflows. Overall, this research demonstrates that developers perceive security can be effectively integrated into Agile development when it is automated, clearly explained, and designed around developers' needs, helping teams build secure software without slowing development.

Dedication

To my family.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Dr. Chris Brown, for his unwavering support, guidance, and mentorship throughout my doctoral studies. His trust in my ideas, thoughtful feedback, and steady encouragement were instrumental in shaping this dissertation and my growth as a researcher. His ability to balance high-level vision with practical guidance helped me navigate both the technical and intellectual challenges of this work.

I am also sincerely thankful to my dissertation committee members—Dr. Na Meng, Dr. Bob Edmison, Dr. Muhammad Ali Gulzar, and Dr. Justin Smith—for their valuable insights, constructive feedback, and thoughtful questions. Each committee member brought a unique perspective that strengthened this research and encouraged me to think more critically about its broader implications and real-world impact.

I would like to thank all the members of the Code-World lab for creating an intellectually stimulating and supportive research environment. The discussions, collaborations, and shared experiences within the lab played an important role in shaping my ideas and sustaining my motivation throughout this journey.

Finally, I am grateful to my family, friends, and colleagues who supported me throughout my doctoral studies. Their encouragement, patience, and belief in me made this long and challenging journey possible.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence

ANOVA Analysis of Variance

AWDWF Agile Web Development with Web Framework

CI/CD Continuous Integration & Continuous Delivery

DAST Dynamic Application Security Testing

IAM Identity and Access Management

IRB Institutional Review Board

LLM Large Language Model

NLP Natural Language Processing

OWASP Open Web Application Security Project

PII Personal Identifiable Information

PR Pull Request

RNN Recurrent Neural Network

SAST Static Application Security Testing

SDLC Software Development Life Cycle

SSO Single-Sign-On

XP Extreme Programming

XSS Cross-Site Scripting

ZAP Zed Attack Proxy

Chapter 1

Introduction

1.1 Problem Definition

Software development practices have evolved significantly, with Agile methodologies emerging as a dominant approach, adopted by over 70% of companies in the United States [1]. Agile emphasizes iterative and incremental development, continuous delivery, and customer satisfaction through collaboration [2]. This methodology represents a departure from traditional software development approaches that relied heavily on extensive planning and documentation [3], offering instead a more flexible and responsive framework for managing changing requirements [4] and customer needs [5].

However, this shift toward Agile methodologies creates a significant challenge in software security. Software security aims to protect data and services so their Confidentiality, Integrity, and Availability (CIA) are preserved and not put at risk [6]. Traditional security engineering, which is inherently sequential in nature [7], does not naturally align with Agile's iterative workflows. Agile's emphasis on flexibility, rapid delivery and customer collaboration [8] could unintentionally lead to security vulnerabilities due to reduced focus on security considerations [9]. This misalignment creates a critical tension between the need for rapid development and robust security measures, particularly in contexts where software handles sensitive information [10] and must comply with data protection requirements [11]. Many Agile frameworks, like Scrum and Extreme Programming (XP), do not inherently prioritize

security, which can result in applications that meet functional requirements but fall short in safeguarding sensitive data [12, 13]. The absence of formal security processes in Agile environments has raised concerns about the long-term viability and safety of software products being developed under these methodologies [14, 15].

Web applications, which are frequently developed using Agile processes [16], increasingly handle sensitive user data [17] and therefore require security testing approaches that can be applied continuously without disrupting development velocity. DAST is used identifying runtime vulnerabilities in web applications [18]; however, such tools tools are often difficult to operationalize in Agile environments due to long execution times [19], brittle CI/CD integrations, and the production of complex, verbose reports that are challenging for developers to interpret and act upon within short iteration cycles [20, 21, 22]. Solution to these challenges would be integrating security practices at every phase of Agile, from inception to deployment [23, 24]. Security practices in software development typically include activities such as threat modeling, secure coding guidelines, security code-reviews, static & dynamic application security testing (SAST & DAST), dependency vulnerability scanning, penetration testing, security knowledge training, etc. [25]. Despite this, there remains limited empirical guidance on how security practices—in particular DAST—can be effectively integrated into Agile workflows in a manner that is both usable for developers and compatible with CI/CD practices. Hence, the importance of effectively addressing this challenge cannot be overstated.

Despite the need to integrate security practices into Agile development, several fundamental challenges continue to hinder the effective adoption of DAST in Agile workflows:

- **Process Integration:** DAST is traditionally applied as a standalone or late-stage security activity, often relying on scheduled scans and centralized security ownership. This model clashes with Agile's rapid, iterative development cycles and continuous

integration practices [26], making it difficult to incorporate DAST into day-to-day development workflows without introducing friction or delays.

- **Tool Complexity:** DAST tools commonly produce complex and lengthy reports that enumerate low-level technical findings without sufficient context or prioritization [27, 28]. In Agile environments with short iteration cycles, developers often lack the time or security expertise required to interpret these reports and translate them into concrete remediation actions, reducing the likelihood that identified vulnerabilities are addressed promptly [22, 29].

- **Practical Implementation:** Although DAST is widely recommended for securing web applications, there remains limited empirical guidance on how DAST can be operationalized within Agile workflows and CI/CD pipelines in practice [30]. Organizations often struggle with questions of when to run scans, how to present results to developers, and how to balance security feedback with development velocity [31, 32].

To address these challenges, there is a need for research that moves beyond high-level discussions of secure Agile development and provides empirically grounded, practice-oriented insights into the integration of DAST within Agile workflows. In particular, such research should examine how DAST impacts Agile development processes in practice and explore concrete mechanisms for making DAST outputs more usable, actionable, and compatible with CI/CD-driven development while maintaining effective protection against evolving security threats.

1.2 Research Contributions

Our research makes the following key contributions to the field of software engineering, specifically focusing on the intersection of DAST security practices and Agile:

1.2.1 Understanding Impact of Security Practices on Agile Development

We conducted a comprehensive survey study (Chapter 4) with 34 software practitioners to investigate how security practices affect various aspects of Agile development, particularly focusing on the challenge of process integration where traditional security practices often clash with Agile's rapid, iterative approach. This study provides valuable insights into practitioners' perceptions of security integration, revealing that while most practitioners acknowledge the benefits of security practices, they express uncertainty about their impact on overall system security and struggle with incorporating security practices into their Agile workflows. These findings offer practical implications for software engineering teams seeking to better integrate security practices into their Agile processes while maintaining development speed.

1.2.2 Enhancing DAST Alert Comprehension through LLM Integration

To address the tool complexity challenge, where security testing tools generate overwhelming and lengthy reports that burden development teams within Agile's time-constrained sprints, we developed an innovative approach using LLMs to summarize security alerts (Chapter 5). Through a survey of 48 software practitioners, we evaluated the effectiveness of

LLM-generated summaries from multiple DAST tools (Burp Suite[1] and ZAP[2]). Our results demonstrate that LLM-generated summaries significantly improve alert comprehensibility, making security findings more accessible to various stakeholders and potentially enhancing overall product security.

1.2.3 Real-World Integration of DAST in Agile Workflows

Addressing the practical implementation challenge, where organizations lack empirical evidence and proven strategies for security integration, we conducted an action-research case study within an Agile (Kanban) team to investigate the real-world integration of DAST into development workflows, deployment pipelines, and day-to-day engineering practices (Chapter 6). Through practitioner interviews and workflow observations, we identified key challenges— understanding security reports, prioritizing security findings, CI/CD pipeline fragility, and reliance on a dedicated security engineer—as well as successful mitigation strategies such as automation, incremental adoption, and embedding scans directly into existing workflows. Our findings demonstrate that DAST can be adopted with minimal disruption when carefully aligned to Agile principles and supported with appropriate tooling.

1.2.4 *SafeAIMerge* — A Tool Integrating DAST and LLM Feedback into GitHub Workflows

Agile methodologies emphasize rapid feedback and iterative development, which are operationalized through CI/CD pipelines that automate integration, testing, and deployment [2, 33, 34]. Building on the previous studies, we designed *SafeAIMerge*, a CI/CD-based security

[1] https://portswigger.net/burp
[2] https://www.zaproxy.org/

tool that combines DAST scanning with LLM-based summarization & remediation to deliver developer-centric security feedback directly within PR workflows (Chapter 7). *SafeAIMerge* embeds automated security analysis into CI/CD pipelines and presents summarized, actionable security insights in the PR context, reducing the need for developers to interpret verbose external reports. We first evaluated practitioner perceptions of *SafeAIMerge* through a formative online survey with 46 participants. The results indicated a strong preference for LLM-generated summaries over traditional DAST reports and highlight the perceived value of integrating security feedback directly into existing development workflows, suggesting high usability and adoption potential. To move beyond perceived usefulness and assess practical impact, we subsequently conducted a controlled, within-subjects summative user study with 12 participants for comparing *SafeAIMerge* against the baseline ZAP workflow. The results demonstrated that *SafeAIMerge* significantly reduced cognitive and emotional workload, enabled faster and more effective vulnerability remediation, and was unanimously preferred by participants. Together, these findings provide rigorous empirical evidence that integrating LLM-assisted, PR-contextual security feedback into CI/CD workflows improves developers' perceived effectiveness and overall experience during security remediation tasks.

These contributions collectively advance our understanding of security integration in Agile environments while providing practical solutions for industry practitioners. Each contribution addresses a specific aspect of the security-agility challenge, offering theoretical insights and practical recommendations for improving security practices in Agile development contexts.

1.3 Dissertation Goal

The goal of this dissertation is to empirically investigate how security practices, particularly DAST, are currently perceived and practiced in Agile software development workflows, and to design and evaluate LLM-assisted mechanisms that improve the usability, actionability and workflow integration of DAST feedback within CI/CD pipelines. Specifically, this research focuses on three interrelated dimensions observed in Agile development settings. First, we examine the *process integration challenge* by investigating how software practitioners perceive and experience the integration of security practices within Agile workflows, and by conducting an industry case study where DAST was integrated into a Kanban-Agile workflow. Second, we tackle the *tool complexity challenge* by improving the usability and actionability of DAST outputs through the use of LLMs. Third, we address the *practical implementation challenge* by designing and empirically evaluating mechanisms for embedding DAST feedback directly into developer workflows and CI/CD pipelines.

Together, these contributions provide both conceptual insight and practical guidance on how DAST can be better adapted to fit Agile environments, reduce cognitive burden on developers, and improve the effectiveness of vulnerability remediation without compromising development agility.

1.4 Dissertation Organization

The following outlines the organization of this dissertation:

- Chapter 2 reviews prior research related to this work, including studies on integrating security practices into Agile development and presents findings from a systematic literature survey on the use of LLMs in cybersecurity.

- Chapter 3 presents background and foundational concepts relevant to this dissertation, including Agile software development processes, security practices in the software development life cycle, DAST methodologies, and LLMs.
- Chapter 4 reports on an empirical survey study examining software practitioners' perceptions of security practices in Agile environments, focusing on perceived benefits, challenges, and impacts on development workflows.
- Chapter 5 investigates the use of LLMs to improve the comprehensibility of DAST alerts and presents the results of a practitioner study evaluating LLM-generated security summaries across multiple DAST tools.
- Chapter 6 describes an action-research case study on the real-world integration of DAST into an Agile (Kanban) development team, detailing observed challenges, mitigation strategies, and practitioner experiences.
- Chapter 7 introduces *SafeAIMerge*, a CI/CD tool that combines DAST scanning with LLM-based summarization, and presents the results of survey-based and controlled user studies evaluating its effectiveness and usability.
- Chapter 8 concludes the dissertation and outlines directions for future research on integrating security practices into Agile software development.

1.5 Thesis Statement

Through LLM-enhanced security tooling and practitioner-centered implementation strategies, security practices can be effectively integrated into Agile development in ways that align with developers' perceptions, workflows, and remediation needs.

Chapter 2

Related Work

2.1 Impact of Security Practices in Agile

Prior research has examined software development practices within Agile environments and challenged the widespread assumption that Agile methodologies inherently compromise software security. Rindell et al. [35] demonstrated that this perception is largely unfounded, showing instead that security-related activities are increasingly being adopted in Agile teams. Consistent findings were reported by Jabangwe et al. [36] in a systematic literature review exploring existing approaches for integrating security practices into Agile development processes. While these studies provide valuable conceptual and theoretical insights, they rely primarily on secondary evidence. In contrast, this work seeks to complement this body of literature by empirically examining security integration from a practitioner-centered and real-world perspective.

Several studies have investigated the compatibility of traditional security practices with Agile methodologies. Beznosov et al. [37] categorized security assurance techniques based on their adaptability to Agile development and found that a substantial portion of established practices are poorly aligned with Agile workflows due to their reliance on extensive documentation and rigid processes. Similarly, Bartsch et al. [38] conducted a literature review on security challenges in Agile projects and emphasized the importance of explicitly addressing non-functional security requirements early in the development lifecycle. They further rec-

ommended incorporating security risk awareness into Agile retrospectives. Building on these insights, our research focuses on practitioners' lived experiences with security integration in Agile workflows and proposes practical, workflow-compatible strategies to address the challenges identified in prior work.

Risk management within Agile development has also been the subject of prior investigation. Hammad et al. [39] examined how risk management practices are applied in Agile projects and found that, although such practices are commonly used, they are often implemented in an ad hoc and non-systematic manner. Their survey-based study identified project deadlines and evolving requirements as the most frequently encountered risks. Our work extends this line of research by shifting the focus from general risk management to the integration of concrete security activities, such as DAST, within Agile workflows, to provide empirical insights that go beyond traditional discussions.

Other research has examined broader limitations of Agile methodologies and their implications for software quality. Agrawal et al. [40] investigated the strengths and challenges of Agile development through an online survey of practitioners, identifying issues such as limited upfront planning, budget constraints, insufficient documentation, and reduced predictability. This work builds on these findings by examining how the integration of security practices further complicates Agile workflows. By doing so, we highlight the need for thoughtful or developer-centric integration strategies that reconcile Agile principles with security requirements.

Empirical studies have also explored how security practices are applied in real-world development settings. Assal et al. [41] investigated software security practices across different stages of the software development lifecycle through in-depth interviews with developers, revealing discrepancies between recommended best practices and their actual adoption. Given the widespread adoption of Agile methodologies, our study narrows this focus to security

practices specifically within Agile development contexts. Similar to Assal et al. [41], we emphasize real-world practices rather than prescriptive guidelines, while additionally examining how security activities influence Agile workflows and team dynamics. Furthermore, we extend prior work by investigating practitioners' perceptions of the effectiveness of security practices in Agile environments and their willingness to adopt and continue using them.

Finally, comparative studies have highlighted the limitations of applying traditional security engineering approaches to Agile projects. Ayalew et al. [42] compared established waterfall-based security engineering processes with Agile methodologies and emphasized the need for security practices tailored specifically to Agile development. They also highlighted the importance of balancing security benefits with the associated integration costs. While such studies primarily focus on identifying theoretical challenges and methodological mismatches, this work adopts a comprehensive and practice-driven approach to addressing security integration in Agile environments. Through empirical investigations, we examine practitioner perceptions of security activities and propose concrete solutions, including LLM-enhanced security report comprehension and evidence-based implementation strategies.

2.2 Web Application Security and Agile

Securing web applications within rapidly evolving development environments has been the focus of extensive research. Scholars have introduced a range of methods aimed at embedding security earlier and more effectively in the software development process, including incremental risk and policy assessments [43], security-oriented lifecycle models [44], approaches for continuous security testing [45], and the broader DevSecOps paradigm [46]. In the context of web application development specifically, frameworks such as Agile Web Development with Web Frameworks (AWDWF) [16] and mappings between secure Scrum processes and

the ISO Systems Security Engineering–Capability Maturity Model (SSE-CMM) [47] have also been proposed. However, these contributions offer limited insight into how automated security analysis tools (e.g. SAST and DAST) can be effectively incorporated into modern development workflows.

Despite broad agreement that integrating security practices throughout development is essential for reducing vulnerabilities and preventing attacks, empirical work shows that doing so in Agile [26, 48] and CI/CD [45, 49] environments remains difficult. Studies have documented developer perceptions of friction when attempting to adopt security measures within highly iterative, automation-driven settings [50]. Rindell et al. [35] found that many practitioners view traditional, phase-based security processes as incompatible with Agile's incremental nature. Bartsch et al. [38], through literature analysis and interviews, reinforced the existence of persistent integration challenges across diverse organizations. Rahman et al. [51] examined online discussions and survey responses, revealing concerns that DevOps activities may deprioritize or negatively influence security tasks.

In addition to technical and process-oriented barriers, prior work highlights the importance of social and collaborative dynamics when embedding security into development work. Ashenden et al. [52] argued that improving software security requires strengthening coordination, trust, and shared understanding between developers and security professionals—especially in environments shaped by open-source collaboration, Agile, DevOps, and DevSecOps. Using social practice theory (SPT), Spotswood et al. [53] studied security as a collective practice rather than an individual responsibility, showing how group norms and interactions shape security behaviors. Through qualitative interviews, the researchers [53] emphasized that co-creation of security activities—such as collaboratively shaping processes, tools, and responsibilities—can improve the adoption and legitimacy of security work within Agile teams. These findings closely align with our work (Chapter 6); however, our study advances

prior research by moving beyond observational and conceptual analyses to operationalize these social dynamics through the integration of DAST practices within a real-world Agile (Kanban) team. We demonstrate how developer buy-in, shared ownership, and collaborative refinement of tooling and workflows directly influenced the sustained adoption of DAST in day-to-day development.

2.3 Developer Behavior and Acceptance of New Practices

Studies grounded in technology acceptance theory [54] demonstrate that developer behavior plays a decisive role in whether practices are adopted, used selectively, or abandoned. Mohagheghi et al. [55], for example, conducted a qualitative multi-case study of Model-Driven Engineering (MDE) [56] adoption across four industrial organizations. Using an extended Technology Acceptance Model (TAM) [54], they found perceived usefulness to be the strongest driver of adoption, while ease of use, tool maturity, and integration with existing workflows posed significant barriers. Organizations adopted MDE cautiously and selectively, emphasizing the importance of workflow compatibility, long-term return on investment, and gradual integration. Similar to our work, this study highlights the importance of developer-centered value and process fit.

Similarly, Senarath et al. [57] conducted task-based survey study of 149 professional developers, investigating intention to follow Privacy Engineering Methodologies (PEMs) [58]. Drawing on TAM, the study showed that perceived usefulness, compatibility with existing practices, and result demonstrability are the strongest predictors of adoption intention. These findings closely inform the design of our *SafeAIMerge* tool evaluation (Chapter 7),

which adopts similar acceptance-oriented indicators in its practitioner surveys.

Riemenschneider et al. [59] reinforced these conclusions through a field study comparing five acceptance models applied to mandated development methodologies. Across all models, perceived usefulness and compatibility with existing work practices emerged as the most consistent predictors of adoption. The researchers argued that developers weigh individual costs against benefits that often accrue at the organizational level, leading to resistance when personal value is unclear. Building on this body of work, our research extends acceptance perspective to security practices—particularly DAST—in Agile workflows, combining empirical studies of developer perceptions with the design and evaluation of tooling interventions—such as LLM-enhanced security feedback—intended to reduce perceived workload, improve comprehensibility, and support efficient remediation of security issues.

2.4 LLMs in Cyber-Security

Recent research has increasingly explored how LLMs can be integrated into various stages of the cybersecurity pipeline. These applications span vulnerability detection and analysis, penetration testing assistance, phishing and malware detection, secure code generation, automated program repair, security operations and threat intelligence, etc. [60, 61]. In many of these settings, LLMs are not proposed as direct replacements for existing security tools, but rather as complementary components that can enhance contextual understanding, reduce noise in tool outputs, and support human decision-making. At the same time, the adoption of LLMs introduces new challenges, including concerns related to hallucinated outputs, interpretability, data privacy, computational cost, and susceptibility to adversarial manipulation [62, 63].

This section records our findings of a systematic literature review we conducted to exam-

ine LLM usage across different cybersecurity tasks, with particular attention to their roles, benefits, and limitations. We followed a structured and reproducible methodology to identify, filter, and synthesize prior research. Relevant studies were collected through systematic searches across major academic databases and digital libraries using combinations of LLM-related and cybersecurity-focused keywords. Clear inclusion and exclusion criteria were applied to retain recent (2020–2025), English-language studies that explicitly employed LLMs in cybersecurity tasks in a novel manner. The finalized set of papers was analyzed qualitatively through an iterative review process by two researchers.

2.4.1 Static Application Security Testing

SAST refers to the automated analysis of source code or binaries to detect potential security vulnerabilities without executing the program [64]. Prior research has shown that traditional SAST tools often suffer from shallow program analysis and high false positive rates, which can reduce their practical usefulness and impose substantial manual triage effort on developers [65]. Recent work has therefore explored the use of LLMs to augment or complement SAST or similar workflows, with the goal of improving vulnerability coverage, contextual understanding, and result accuracy.

Keltek et al. [66] proposed *LLM-supported Static Application Security Testing (LSAST)*, a framework in which SAST findings are used as structured input to an LLM that reasons about potential additional vulnerabilities. The study demonstrated that fine-tuned LLMs can improve vulnerability detection capabilities when paired with static analysis outputs, but also highlighted the challenges related to static training data, privacy concerns, and the risk of outdated vulnerability knowledge. To mitigate these issues, the authors advocate for locally hosted LLMs and retrieval-based mechanisms to continuously incorporate recent

security information. Similarly, Lekssays et al. [67] proposed *LLMxCPG*, a context-aware vulnerability detection approach that combines code property graphs (CPGs) with LLMs to extract vulnerability-relevant program slices before classification. By using LLM-generated CPG queries to reduce the input while preserving security-critical dependencies, their results suggest improved robustness and scalability compared to applying LLMs directly to raw code.

Complementary findings are reported by Shashwat et al. [68], who conducted a preliminary study comparing LLM-based source code analysis against SonarQube[1] on OWASP[2] benchmark applications. Their results indicated that LLMs can achieve detection accuracy comparable to, and in some cases exceeding, that of traditional SAST tools for specific vulnerability types. At the same time, the study observed that LLMs tend to introduce additional false positives, underscoring the importance of careful prompt design and iterative refinement.

While some studies focus on improving vulnerability recall, others emphasize the role of LLMs in reducing false positives generated by SAST tools. Wagner et al. [69] investigated the use of LLMs to reassess and validate SAST findings produced by tools such as SpotBugs.[3] Experimental results showed that LLMs can substantially reduce false positive rates while maintaining high true positive coverage, although performance varies across models. The study also highlighted important limitations, including hallucinated reasoning, computational costs, and the need for explanations that developers can trust.

Beyond function-level analysis, Zhou et al. [70] examined the effectiveness of LLMs and SAST tools for repository-level vulnerability detection across Java, C, and Python projects. Their large-scale comparison of 15 SAST tools and 12 state-of-the-art LLMs revealed a clear trade-off: LLMs achieve significantly higher true positive rates, often detecting 90–100%

[1] https://www.sonarsource.com/products/sonarqube
[2] https://owasp.org/www-project-benchmark/
[3] https://spotbugs.github.io/

of vulnerabilities, but at the cost of substantially higher false positive rates compared to traditional SAST tools. The study introduced a new dataset for repository-level vulnerability detection and demonstrates that combining SAST outputs with LLM reasoning can significantly improve detection ratios while reducing the number of flagged functions.

Focusing on developer workflows, Steenhoek et al. [71] evaluated an IDE-integrated LLM-based vulnerability detection and repair tool with 17 professional developers using their own projects, focusing on real-world usefulness, trust, and workflow fit. Their findings highlight that false positives and non-applicable fixes remain major barriers in practice, often driven by missing program/environment context and limited customization to project-specific conventions, underscoring the importance of deployment-oriented evaluations beyond benchmark performance.

Taken together, these studies suggest that LLMs offer strong contextual reasoning capabilities that can enhance static vulnerability detection, particularly in identifying complex or cross-cutting security issues that challenge traditional SAST tools. However, they also reveal recurring limitations, including elevated false positive rates, dependence on prompt quality, data freshness concerns, and deployment challenges related to privacy and computational cost. As a result, the literature increasingly converges on hybrid architectures in which LLMs are used to augment, validate, or contextualize SAST outputs rather than replace existing tools. These findings motivate further exploration of LLM-assisted static analysis approaches that balance detection coverage with precision while remaining practical for integration into real-world development workflows.

2.4.2 Automated Penetration Testing

Penetration testing represents a more complex and interactive security task than static or dynamic analysis, requiring iterative reasoning, automated testing tools such as fuzzing, and the ability to maintain context across multiple exploitation steps [72]. Fuzzing, or fuzz testing, is an automated software testing technique designed to discover security vulnerabilities by sending a large volume of random, malformed, or unexpected inputs to an application [25]. Recent work has explored whether LLMs can support or automate such workflows by acting as task-planning agents.

Deng et al. [73] introduced *PENTESTGPT*, an LLM-driven system designed to automate penetration testing by decomposing high-level security objectives into executable sub-tasks. The tool was evaluated on the picoMini Capture-the-Flag (CTF) benchmark across categories including web, forensics, cryptography, and binary exploitation. Their results showed that *PENTESTGPT* adopted problem-solving strategies similar to those of human penetration testers, effectively prioritizing sub-tasks rather than reacting only to the most recent finding. In comparative evaluations, the approach significantly outperformed baseline models, achieving a 228.6% improvement in task completion over GPT-3.5.

Despite these promising results, the study also highlighted important limitations of LLM-based penetration testing. The system frequently exhibited depth-first search behavior, leading to excessive focus on individual services while neglecting previously identified attack paths. Additional challenges included hallucinated or inaccurate command generation and difficulty maintaining long-term memory required to link vulnerabilities and develop coherent exploitation strategies. These findings suggest that while LLMs show potential as assistants or planners in penetration testing workflows, reliable automation of end-to-end penetration testing remains an open challenge.

2.4.3 Automated Threat Detection and Prevention

Automated threat detection and prevention encompass a broad set of defensive activities, including identifying malicious artifacts (e.g., phishing URLs, malware traces, and malicious packages), extracting actionable intelligence from security data, and enabling timely mitigation through analysis and response. Recent work increasingly positions LLMs as adaptive components within these pipelines because they can process heterogeneous inputs (natural language, code, logs, and semi-structured artifacts) and generalize to evolving threat patterns [60].

A substantial portion of recent literature focuses on phishing detection across URLs, webpages, email, and SMS. Wang et al. [74] proposed an agentic phishing detection framework that performs online information retrieval to mimic human decision-making, leveraging external resources (e.g., Google Search and logo detection) to improve brand recognition and classification. Their results suggested notable gains over prior systems, but also exposed operational limitations such as increased runtime cost and sensitivity to long HTML pages that constrain the number of tool interactions. Complementing agentic designs, Trad et al. [75] compared prompt engineering and fine-tuning for website phishing detection using a large URL dataset, showing that fine-tuned models substantially outperform prompting-only baselines and classical ML approaches, while also demonstrating that prompt templates (e.g., role-based and chain-of-thought prompts) can materially influence accuracy. Mahendru et al. [76] examined phishing detection across multiple modalities and compare LLMs against DeBERTa-based models, reporting that DeBERTa can outperform GPT-4 on recall for email phishing, while LLMs exhibit advantages in detecting synthetic or newly emerging phishing patterns; their findings further highlighted that synthetic data can improve robustness against LLM-generated phishing attacks.

LLM-based approaches also appear in malware-focused pipelines, both for detection and for modeling attacker adaptation. Sanchez et al. [77] proposed a transfer-learning framework that applieed pre-trained language models to system-call traces for malware detection in a military setting, emphasizing the scale of operational data and reporting that larger context sizes can improve classification performance, while also noting that computational overhead can limit real-time applicability. In contrast, Hu et al. [78] introduced *MalGPT*, a causal language model designed to generate malware variants that evade deep-learning malware detectors, illustrating how generative models can be used to emulate adversarial malware example generation and stress-test defensive systems. Together, these studies reinforce the dual-use dynamic of LLMs: the same modeling capabilities that support defensive detection can also enable more effective evasion and attack automation.

Beyond phishing and malware binaries, software supply chain threats have received growing attention. Zahan et al. [79] evaluated OpenAI's[4] GPT-3 and GPT-4 for detecting malicious `npm` packages and proposes *SocketAI*, a workflow that combines static analysis pre-screening with LLM-based review. They report that LLMs can outperform *CodeQL* for malicious package detection and that hybrid pre-screening substantially reduces the number of files requiring LLM analysis, improving cost-effectiveness and scalability. Complementing this line of work, Sun et al. [80] addressed vulnerability traceability in supply chains by using LLM-assisted repository identification to map reports to vulnerability-relevant files, and showed through a user study with eight participants, that such support improves both speed and correctness in locating affected code. Related prevention-oriented work also targets vulnerability hunting in web applications: Sakaoglu et al. [81] proposed *KARTAL*, which combines fuzzing-derived behavioral traces with prompt construction and a fine-tuned LLM detector to identify logical web vulnerabilities (e.g., broken access control), and reported

[4] https://openai.com/

that performance improves as dataset size increases, while also acknowledging constraints such as sequence length limits, training cost, and manual labeling effort.

Several studies further frame "prevention" as improving the ability to identify and address vulnerabilities before exploitation, spanning detection, localization, secure generation, and repair. For vulnerability detection, Shestov et al. [82] studied fine-tuning LLMs for Java vulnerability detection and highlight challenges related to highly imbalanced real-world data and the compute required for full-model tuning. Wang et al. [83] proposed *DefectHunter* and reported improved vulnerability identification on mixed datasets, while Gonçalves et al. [84] introduceed *SCoPE*, emphasizing dataset normalization and deduplication for C/C++ vulnerability detection in IoT-relevant settings and observed that performance on real-world code can remain limited despite pre-processing.

Retrieval augmentation is also explored to improve reliability: Du et al. [85] proposed *Vul-RAG*, which constructs a CVE-derived vulnerability knowledge base and retrieves functional-semantics-aligned knowledge to guide detection and reasoning, reporting improved accuracy and precision–recall trade-offs. Similarly, Mathews et al. [86] showed that prompt engineering and RAG-style contextual support can improve Android vulnerability detection accuracy, while noting risks of prompt bias and outdated model knowledge. Zhang et al. [87] further demonstrated that enhanced prompting with auxiliary program information (e.g., data flow and API calls) can significantly affect detection performance and that these gains vary across programming languages. In smart contract analysis, Ince et al. [88] found that fine-tuned open-source models can achieve competitive detection performance relative to proprietary models while trading off speed and false positives against static and dynamic analyzers.

Finally, predictive threat intelligence has been explored as a way to move from reactive detection to proactive defense. Bokkena et al. [89] proposed a Predictive Threat Intelligence Model (PTIM) trained on large-scale heterogeneous security data (e.g., logs, phishing emails,

malware samples, and incident reports) and reported improved accuracy, reduced false positives, and faster processing compared to traditional systems, while also emphasizing the importance of transparency, computational feasibility, and privacy-aware deployment.

Overall, prior work suggests that LLMs contribute to automated threat detection and prevention in three recurring roles: (i) adaptive classifiers and analyzers for phishing and malware detection [74, 75, 76, 77], (ii) retrieval-augmented and agentic systems that ground decisions in external evidence or structured knowledge [74, 85, 86], and (iii) prevention-oriented developer workflows that connect detection with localization. However, deployment risks—including adversarial misuse, hallucination, privacy leakage, and operational cost—remain central barriers to dependable real-world adoption [60, 78, 90, 91].

2.4.4 Secure Code Generation and Automated Program Repair

Recent work has increasingly examined whether LLMs can not only generate syntactically correct code, but also produce code that is secure by design and capable of repairing existing vulnerabilities. This line of research is motivated by evidence that code generated by general-purpose LLMs may inadvertently reproduce insecure patterns present in their training data, raising concerns about their reliability in security-critical software development contexts.

Wang et al. [92] provided a systematic evaluation of the security of AI-generated code through *CodeSecEval*, a benchmark designed to assess security awareness during code generation and repair tasks. Their results show that while LLMs can outperform rule-based static analyzers for certain vulnerability categories, they frequently overlook others, and even state-of-the-art models often generate insecure code. These findings highlight that strong functional performance does not necessarily translate to secure outputs and that current LLMs struggle with specific classes of vulnerabilities.

To address these limitations, several studies explore security-centric fine-tuning strategies. He et al. [93] introduced *SafeCoder*, which applies supervised instruction tuning using curated security datasets and demonstrates substantial improvements in secure code generation, though challenges related to data imbalance remain. Similarly, Li et al. [94] investigated parameter-efficient fine-tuning techniques such as LoRA and IA3 on real-world C/C++ datasets, reporting consistent but modest improvements that vary by vulnerability type and code granularity.

Beyond code generation, LLMs have also been applied to automated program repair (APR). Li et al. [95] showed that parameter-efficient fine-tuning can significantly improve repair effectiveness while reducing computational cost compared to full-model fine-tuning and traditional APR techniques. Several works further emphasized grounding LLM-based repair in external context: *InferFix* [96] combined static analysis with retrieval-augmented prompting to guide patch generation using semantically similar historical fixes, while Zhao et al. [97] incorporated design rationales extracted from issue logs to improve patch quality. More recent conversational approaches, such as *ContrastRepair* [98], demonstrated that iterative interaction with contrastive test cases can further improve repair accuracy while reducing the number of model interactions.

Overall, the literature suggests that while off-the-shelf LLMs are not yet reliable for secure code generation or repair, targeted fine-tuning, retrieval augmentation, and integration with static analysis and testing artifacts can substantially improve security outcomes. At the same time, recurring challenges remain, including vulnerability-specific performance variation, data imbalance, evaluation bias, and the difficulty of providing LLMs with sufficient project-wide context. These findings indicate that secure code generation and repair are most promising when LLMs are embedded within hybrid, tool-supported workflows rather than used as standalone code generators.

2.4.5 Summary

Table 2.1: Summary of included studies grouped by cybersecurity application area.

Area	**Studies**
SAST	[66, 67, 68, 69, 70, 71]
Penetration testing	[73]
Automated threat/vulnerability detection and prevention	[74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 99]
Secure code generation and automated program repair	[92, 93, 94, 95, 96, 97, 98]

We saw how LLMs are being used in cybersecurity, focusing on how they are being applied across different security tasks (see Table 2.1) and what benefits and limitations have been observed in practice. Rather than treating LLMs as monolithic solutions, the literature consistently frames them as complementary components that augment existing security techniques through improved contextual reasoning, adaptability, and human-centric interaction. However, several important research gaps remain. In particular, there is limited empirical evaluation of developer perceptions of LLM-generated security feedback. In addition, prior work has paid comparatively little attention to the application of LLMs within specific security practices such as DAST, as well as to their integration into real-world developer and operational workflows. Together, these gaps motivate continued investigation into LLM-assisted security systems that are both effective and trustworthy, providing the foundation for the research contributions presented in the subsequent chapters.

Chapter 3

Background

3.1 Software Security

Software security encompasses the principles, practices, and tools designed to protect systems from malicious attacks while safeguarding user data integrity and privacy. The critical nature of security in modern software development is exemplified by high-profile security breaches that have resulted in substantial financial and reputational damages. For instance, the 2017 Equifax data breach—involving one of the largest credit reporting agencies in the U.S.—resulted from a vulnerability exploitation that exposed sensitive PII of approximately 147 million users [100]. The breach's aftermath included billions in lost market value and a $700 million settlement, underscoring why organizations must prioritize security integration throughout their software development lifecycle.

The complexity of securing modern software systems presents mounting challenges for practitioners [101]. These challenges stem from several factors:

- Increasingly sophisticated cyber threats and attack vectors
- Growing complexity of software architectures and dependencies
- Integration of diverse third-party components and services
- Pressure to maintain rapid development and deployment cycles

- Balance between security requirements and user experience

To address these challenges, the software industry has developed numerous automated tools and methodologies to support security-related tasks. These range from automated vulnerability scanners to secure coding frameworks and security testing tools.

3.2 Dynamic Application Security Testing (DAST)

DAST represents a black-box security testing methodology designed to evaluate web applications and APIs during runtime [102]. In contrast to SAST, which analyzes source code directly [64], DAST approaches security testing from an external perspective, simulating potential attacker behavior. This approach identifies various security vulnerabilities such as SQL injection[1], cross-site scripting (XSS)[2], etc., by interacting with the application in its running state [81]. DAST is distinguished by several fundamental characteristics:

- **Black-box Testing:** DAST employs black-box testing methodologies [103], operating without access to the application's source code and instead focusing on external interfaces such as web forms and APIs [81].
- **Runtime Analysis:** The dynamic nature of DAST enables real-time application analysis through systematic request generation and input testing, revealing vulnerabilities that may only emerge under specific operational conditions [103].
- **Attack Simulation:** DAST tools simulate real-world attack scenarios [104] by deploying malicious payloads and conducting systematic vulnerability probing, providing organizations with practical insights into their security posture.

[1] SQL Injection
[2] Cross-Site Scripting (XSS)

Organizations can choose from various DAST solutions, including commercial and open-source options. This work focused on two widely adopted tools: BurpSuite, a comprehensive commercial security testing platform, and ZAP, a popular open-source security scanner:

ZAP Zed Attack Proxy (ZAP) is a free, open-source cross-platform web application security scanner [105] which is supported by Checkmarx[3]. ZAP combines automated vulnerability scanning capabilities with an interactive environment for manual testing and analysis.

Burp Suite Developed by PortSwigger[4], Burp Suite offers comprehensive web vulnerability scanning capabilities. It provides both automated and manual testing functionalities, featuring specialized tools such as Proxy, Intruder, Repeater, etc., for advanced penetration testing [106].

Upon completion of application scanning, DAST tools generate detailed vulnerability reports categorized by severity levels. Examples of these security reports generated by both Burp Suite and ZAP can be accessed through our public repository.[5]

3.3 Large Language Models (LLMs)

LLMs have emerged as revolutionary tools in Natural Language Processing (NLP) and Artificial Intelligence (AI), fundamentally transforming how machines process and generate human language [107, 108]. These sophisticated machine learning models demonstrate remarkable capabilities in understanding, generating, and manipulating natural language [108], representing a significant advancement in AI technology.

[3] https://checkmarx.com/
[4] https://portswigger.net/
[5] Sample security reports from ZAP and Burp Suite

The evolution of LLMs is closely tied to developments in deep learning architectures, particularly the advancement of Recurrent Neural Networks (RNNs) and transformer models [109]. The realization of powerful LLMs, such as GPT-3, has been made possible through the convergence of enhanced computing infrastructure and extensive training datasets [110].

LLMs are distinguished by several key attributes:

- **Architectural Scale:** Modern LLMs are characterized by their unprecedented scale, incorporating hundreds of millions to billions of parameters [111]. This massive scale enables the capture of complex language patterns and contextual nuances, leading to more precise and contextually appropriate outputs.

- **Training Methodology:** These models undergo extensive training on large-scale text corpora, requiring significant computational resources [112]. Post-training, they can be further refined through fine-tuning processes for specific applications or domains, enhancing their task-specific performance [113].

- **Application Versatility:** LLMs demonstrate remarkable versatility across various NLP applications such as language translation [114], text summarization [115], sentiment analysis [116], and question answering systems [117]. This flexibility has led to their adoption across diverse sectors, from healthcare and customer service [118] to financial services and entertainment industries [119].

- **Adaptive Learning:** Advanced LLMs incorporate continual learning capabilities, enabling ongoing adaptation and improvement through exposure to new data and tasks [120]. This feature ensures their sustained effectiveness in dynamic environments where language patterns and usage continuously evolve.

3.4 Agile

Software engineering processes structure development activities within the SDLC. The SDLC encompasses requirements analysis, system design, implementation, testing, maintenance, and deployment phases. Agile methodologies emphasize iterative development, continuous delivery, and customer-focused collaboration [121]. These methodologies aim to accommodate change adaptively, minimizing increased development costs and unnecessary rework typical in traditional SE approaches [122].

Recognized for its lightweight, adaptable nature [123], Agile was formalized through the 2001 "Manifesto for Agile Software Development"[6], establishing core development values [124]. These principles demonstrate varying implications for software security implementation:

- **Individuals and interactions over processes and tools:** The emphasis on interpersonal communication can enhance security vulnerability awareness and mitigation strategies. Research indicates security tool adoption often occurs through peer recommendations [125], while peer code reviews effectively identify potential vulnerabilities [126]. However, security implementation frequently requires tool and library adoption, presenting usage challenges for developers addressing vulnerabilities [28, 127].

- **Working software over comprehensive documentation:** While prioritizing functional software can validate system security through implementation, security activities generate substantial documentation regarding potential vulnerabilities. Security tools produce issue reports, increasing documentation requirements [128]. Documentation minimization risks inadequate security knowledge preservation, while complex security tool outputs impede adoption in DevOps [129] and development environments [28].

[6] https://agilemanifesto.org/

- **Customer collaboration over contract negotiation:** Early customer engagement in security discussions effectively addresses security requirements [130]. However, clients may lack security implementation expertise or risk assessment capabilities. Formal arrangements often become necessary for security practices, such as external security audits and red team assessments, commonly employed by major software organizations like Microsoft [131].

- **Responding to change over following a plan:** Agile's adaptability [122] enables rapid security incident response, reducing vulnerability remediation timeframes [132]. However, security integration often requires structured approaches. Traditional security frameworks like Microsoft's Security Development Lifecycle [133] face scaling challenges in Agile environments due to iterative development constraints [134]. Security standards implementation, such as FIPS, necessitates documented security planning for system information management [135].

Implementation success varies based on individual team approaches to integrating these values with security practices. This research examines practitioner experiences and perspectives regarding security practice integration within Agile development frameworks.

3.4.1 Kanban

Kanban represents a prevalent Agile methodology [136] characterized by workflow visualization through card systems [137], work-in-progress (WIP) limitations [138], and continuous delivery implementation without rigid temporal constraints typical of other Agile frameworks like Scrum[139] . Derived from lean manufacturing principles [140], Kanban optimizes efficiency by enabling capacity-based task allocation [141] rather than sprint-based scheduling. This flexibility particularly benefits teams requiring adaptive workload and priority manage-

ment [142]. The methodology utilizes board structures segmented into developmental stages (e.g., “To Do,” “In Progress,” “In Review,” “Done”) [143]. This pipeline approach facilitates continuous feature deployment, creating dynamic development environments. WIP limitations prevent team overload [144], reducing process bottlenecks and enhancing workflow efficiency.

3.5 Continuous Integration and Continuous Delivery (CI/CD)

CI/CD represents fundamental practices in modern software development, enhancing project efficiency through automation [145]. These practices operate at different levels of automation, forming a comprehensive pipeline for software delivery. In Continuous Integration (CI), developers frequently integrate code changes into a shared repository, triggering automated builds and tests to ensure code quality and prevent integration issues [146]. Continuous Delivery (CD) extends CI by automating the deployment process across various environments. After successful CI verification, the code progresses through additional testing phases, potentially including integration testing, user interface validation, and security testing. While Continuous Delivery typically requires manual approval for production deployment, its variant, Continuous Deployment, automates the entire process from code commit to production release [146].

Modern CI/CD pipelines have several stages: code commit, automated build, unit testing, and deployment to testing environments [147]. Organizations increasingly incorporate security testing stages into these pipelines, particularly in DevSecOps practices [45], though such testing traditionally wasn’t part of standard CI/CD workflows. This evolution in pipeline

design reflects the growing emphasis on integrating security practices earlier in the development cycle.

Chapter 4

Securing Agile: Assessing the Impact of Security Activities on Agile Development

4.1 Motivation

Over the past decade, Agile development approaches have gained significant traction and are now broadly adopted across software engineering teams. Prior studies indicate that Agile practices can improve team efficiency and software quality [148, 149], while also promoting collaboration and the exchange of knowledge among team members [148, 149]. However, other research has raised concerns about the implications of Agile processes for software security, highlighting potential tensions between Agile principles and security requirements [37, 150]. For example, Agile's emphasis on minimal documentation, which contrasts with the outputs of many security-focused tools—such as code analysis and vulnerability scanners—that generate extensive documentation to describe identified risks and remediation guidance. To address these challenges, researchers and practitioners have proposed incorporating explicit security activities into Agile workflows [15, 134]. In this context, DevOps—a development paradigm often used alongside Agile methodologies [151]—prioritizes automation across development and infrastructure to enable frequent and reliable

software releases [152]. Building on this foundation, *DevSecOps* expands the DevOps model by embedding security considerations throughout the pipeline, for example by integrating automated static analysis and other security tools into continuous integration and deployment processes [46, 153]. Despite these advances, there remains limited empirical understanding of how such security activities align with Agile values, influence day-to-day development practices, and how they are perceived by Agile practitioners. This work seeks to address this gap by examining software practitioners' experiences and perspectives on integrating security activities within Agile development processes. We examine how practitioners integrate security and agile methods through a survey of 34 software professionals, contributing empirical evidence for understanding current industry practices.

Our analysis began with a structured examination of prior academic literature at the intersection of Agile software development and software security. We surveyed peer-reviewed conference and journal publications from major software engineering venues, including ICSE, FSE, ASE, IEEE, and ACM journals. The review targeted studies that explicitly discussed security challenges in Agile environments, proposed security-focused practices or activities, or reported empirical observations of security integration in iterative development processes. From this, we extracted and synthesized recurring security activities, with particular emphasis on practices that were compatible with iterative, fast-paced workflows and feasible for adoption by Agile teams. Through iterative coding and consolidation of overlapping concepts, we identified eight security activities that demonstrated strong potential for strengthening security within Agile development workflows. These activities are summarized in Table 4.1. We aimed to understand software practitioners' perspectives and experiences integrating security activities into Agile practices. This study explored the following research questions (RQs):

RQ1 How do software practitioners perceive the effectiveness of adopted and state-

of-the-art security practices, and what is their level of willingness to incorporate them into the Agile software development process?

RQ2 How are the team velocity and productivity, as perceived by the software practitioners, affected by the inclusion of security activities?

RQ3 What is the impact of integrating security activities into Agile development on software practitioners' confidence in their software product and organization?

Through investigation of these research questions, the results indicate that while security activities are not universally adopted in Agile development, they are generally perceived as effective and valuable when integrated into existing workflows. Practitioners reported particularly strong perceived effectiveness and willingness to adopt automated and iterative security activities, which were viewed as well aligned with Agile principles and continuous development pipelines. These findings suggest that automation plays a critical role in reducing friction between security practices and Agile values.

Importantly, the results further indicate that incorporating security activities does not substantially undermine team productivity. Most practitioners reported minimal impact on perceived sprint velocity, with any short-term overhead often outweighed by improvements in software security and increased confidence in both the product and development process. By examining practitioners' adoption patterns, perceived effectiveness, and perceived operational impact of security practices—particularly with respect to sprint *velocity* [154]—this study provides empirical insight into how Agile practitioners perceive the integration of security measures within their development workflows. Overall, the findings suggest that security integration, when supported by automation and actionable feedback, can coexist with Agile practices and inform the design of workflow-compatible security solutions.

Table 4.1: Security Activities for Agile Software Development

Studies	Security Activity	Definition
[155, 156, 157, 158]	Addressing security in early iterations with requirements and testing	This security activity emphasizes the importance of development teams addressing security issues and concerns early in the project before deploying the software.
[155, 157, 159]	Stating security requirements that are expected in the production software	This requires incorporating security expectations in project requirements when describing the responsibilities and behavior of the software.
[158, 159, 160]	Adding a security specialist to your team	Security specialists, such as a Security Master, are members of a development team that focus on security aspects of the project to address concerns and ensure the security of the system.
[47, 134, 161, 162]	Additional points or weights to issues with an impact on security	This activity involves increasing the weights, such as story points in an Agile development environment, of issues that will have a higher impact the security of the product to prioritize security-related tasks and encourage more secure development and testing.
[64, 134, 155]	Iterative and incremental vulnerability and penetration testing	This security activity suggests incorporating recurring security scanning, such as Dynamic Application Security Testing (DAST), to test for security flaws in the working software automatically.
[64, 134, 158]	Iterative and incremental security static analysis	Similar to DAST, Static Application Security Testing (SAST) involves using security-related static analysis tools to detect potential security vulnerabilities by scanning the source code.
[43, 64, 163]	Iterative and incremental risk analysis, countermeasure graphs	This security activity consists of using tools to monitor networks, applications, and infrastructure and perform risk analysis to identify vulnerabilities. These tools can evaluate the system's security and suggest methods to prevent attacks.
[41, 64, 155]	Automatic testing	This security activity involves incorporating secure coding practices, such as vulnerabilities analysis and risk assessment, into the deployment pipeline for software projects. This allows security checks to be automatically triggered with code changes and issues to be addressed before the software is deployed to users.

4.2 Methodology

4.2.1 Data Collection

To gather our research data, we designed and implemented an online survey through QuestionPro[1]. This survey sought to address our research questions regarding how security practices influenced different components of Agile development processes. The questionnaire comprised nine multi-part items exploring these relationships. Before initiating the data collection phase, we obtained approval for the survey and methodology from the Institutional Review Board (IRB).

4.2.2 Participant Recruitment

Our participant selection process prioritized demographic diversity to capture a broad spectrum of viewpoints. The recruitment approach encompassed multiple communication channels to reach potential respondents. We distributed our survey through LinkedIn[2] using targeted invitations and public posts to engage software industry professionals. Additionally, we leveraged Slack[3] to connect with IT practitioners. The recruitment effort also extended to Virginia Tech graduate students who possessed substantial industry experience in professional software development roles.

[1] https://www.questionpro.com/
[2] https://www.linkedin.com/
[3] https://www.slack.com

4.2.3 Survey Structure

The survey gathered demographic data and background information regarding participants' Agile development experience and security implementation knowledge. Respondents provided details about their teams' adopted security activities and evaluated the security practices outlined in Table 4.1. This set of eight security practices were derived from prior works [26, 36] that conducted a comprehensive literature review on security challenges and mitigation strategies in Agile software development. Specifically, we included the security practices identified and synthesized in that review as representative activities that have been repeatedly recommended for improving security in Agile and CI/CD-oriented workflows. By grounding our survey in this established body of work, we ensured that the selected practices reflect widely recognized and empirically discussed approaches within the Agile security literature. Given our research emphasis on security protocols within Agile development frameworks, we excluded responses from participants that lacked Agile experience.

The questionnaire explored participants' views on integrating security activities into Agile methodologies, examining the impact of these practices on individual workflows, team dynamics, and organizational outcomes. To address RQ1, we examined participants' assessments of each security practice's individual effectiveness in enhancing overall software security. The survey also measured participants' willingness to incorporate these established security practices into their Agile development processes. For RQ2, we investigated how these security activities influenced sprint velocity. To address RQ3, we assessed participants' confidence levels regarding their software products' overall security.

Notably, our survey design incorporated optional response sections, which resulted in a 41% response rate ($n = 14$) for certain questions. These optional inquiries explored the multifaceted effects of security practices within Agile frameworks, examining their influence

on team dynamics, productivity metrics, software quality, organizational performance, and daily operational activities. While optional, these responses provided significant insights into the integration of security practices within Agile methodologies. The complete survey and its structural format is available in the Appendix A.

4.2.4 Data Analysis

The survey incorporated multiple question formats: closed-ended, Likert scale, and open-ended inquiries examining participants' perspectives on software security implementation and its integration with Agile methodologies. Our analytical framework employed a mixed methods approach for evaluating the 5-point Likert scale responses.

Statistical analysis included a one-way Analysis of Variance (ANOVA) to examine response variations across activities, with statistical significance established at $p < 0.05$. For qualitative data analysis, we implemented an iterative-inductive thematic open coding methodology. The analysis involved two researchers conducting independent open coding assessments, followed by collaborative discussions to establish consensus on interpretations.

To illustrate our analytical approach, we examined responses regarding security practices implemented in participants' organizations. The open coding process identified key phrases indicating specific security activities. For instance, in the response *"Dual-authentication, least privilege so only certain users could access certain stage environments just as testing"*, the phrase "Dual-authentication" corresponded to *Multi-Factor Authentication* practices. Similarly, *"Only certain users could access certain stage environments"* indicated implementation of *Identity Access Management* protocols. As demonstrated in Table 4.2, each response underwent systematic parsing and categorization into relevant security practices. This methodical coding approach revealed recurring patterns and themes, enhancing our

understanding of participants' experiences with security integration in Agile development frameworks. The analysis yielded insights into diverse security activities employed within Agile software development environments.

Table 4.2: Open coding examples for "What are the security practices used in your Agile process?"

Participant	Response	Categories Identified
P1	"Security training which takes place each quarter. And quiz based on that training."	Security Training
P3	"Use of continuous Integration, Add multiple checks for static analysis and code scan Proper IAM policy."	Continuous Integration (CI), Static Application Security Testing (SAST), Security scanning, Identity Access Management (IAM)
P4	"Compliance best practices and checklist"	Compliance best practices
P8	"Differential Access to resources"	Identity Access Management (IAM)
P10	"We follow Owasp standards. Each enhancement goes through the tests according to OWASP standards."	OWASP standards
P12	"In my previous company, there was a separate team that worked on the security aspects of the product, like login, user management, permissions, etc. To prevent unauthorised access to certain parts of software. There were security expectations in each user story, and those aspects used to get tested by QA before deploying the code."	Separate Security team, Identity Access Management (IAM), Agile Stories, Separate security team, Agile stories
P18	"Regular security scans using commercial tools, as well as requested scans of servers and applications by the IT security office. Monitoring of known security outlets for zero-day exploits."	Continuous Integration (CI), Security scanning, Automation tools, Monitoring
P19	"Periodic reviews"	Periodic reviews
P27	"Before any story is picked up, there's a Security Review of the entire epic with the Security Team."	Agile stories, Separate security team, Periodic reviews
P30	"Dual-authentication, least privilege so only certain users could access certain stage environments just as testing."	Multi-factor Authentication (MFA), Identity Access Management (IAM)
P31	"Code review, multiple release plan, security checks by software development engineering team and security team."	Code reviews, Separate security team, Continuous Integration (CI)
P32	"There are security experts to ensure secure practices and regular security testing and analysis is performed."	Security expert, Security practice insurance, Continuous Integration (CI)
P33	"Secure Clouds and systems - MFA - Including a round of security test with every PR or Features - Zero Trust Policies."	Multi-factor Authentication (MFA), Zero trust policy, Secure cloud services & systems, Code reviews

4.2.5 Participants

The study attracted 34 respondents who possessed an average of seven years of software engineering expertise. The participant pool primarily consisted of experienced software engineers from the technology industry. The demographic composition included 67% ($n = 23$) industry professionals, who averaged eight years of technical work experience, while 33% ($n = 11$) were graduate-level university students. Professional participants represented di-

verse organizations including Acquia,[4] Flexcar,[5] GlobalLogic,[6] Decisions,[7] Cvent,[8] Lutron Electronics,[9] Palo Alto Networks,[10] and Microsoft,[11] occupying positions such as software engineer, infrastructure engineer, senior product manager, technical consultant, systems architect, and database administrator. The graduate student cohort demonstrated an average of two years of professional software engineering experience. Survey responses indicated that participants predominantly regarded security as extremely ($n = 25$, 76%) or very important ($n = 7$, 21%) to their software teams. Comprehensive demographic information for the study participants is detailed in Table 4.5.

4.3 Results

4.3.1 RQ1: Security Activities in Agile, Perceived Effectiveness & Willingness to Adopt

Adoption

The survey revealed that 97% of participants ($n = 33$) employed Agile software development methodologies, though only 72% ($n = 23$) integrated security-related activities into their Agile workflows. Analyzing the non-student subset of participants indicated that 77% ($n = 27$) of industry professionals incorporated security activities into their development processes. The participants' primary focus centered on Agile development rather than specialized Agile

[4] https://www.acquia.com/
[5] https://www.flexcar.com/
[6] https://www.globallogic.com/
[7] https://decisions.com/
[8] https://www.cvent.com/
[9] https://www.lutron.com/us/en
[10] https://www.paloaltonetworks.com/
[11] https://www.microsoft.com/

security implementations. Security practices identified within these Agile teams encompassed security training initiatives, monitoring and scanning systems, static analysis tooling, code review protocols, implementation of OWASP standards [164], Identity Access Management systems, Multi-Factor Authentication protocols, zero trust architectures [165], and dedicated security teams.

Perception

A subsequent inquiry examined participants' viewpoints regarding their teams' implemented security practices. The collected responses demonstrated a spectrum of perspectives. The majority characterized these practices as "good" ($n = 8$), "informative" ($n = 2$), "necessary" ($n = 2$), with some noting compliance requirements ($n = 1$). Contrasting opinions emerged from participants who described these activities as time-intensive ($n = 1$) or unfavorable ($n = 1$). Some respondents ($n = 4$) highlighted opportunities for enhancement, underscoring the significance of strengthening existing security protocols. The findings suggested that despite widespread recognition of security practices' value, substantial potential remained for refinement and optimization within development teams.

Agile Security

The survey examined participants' agreement levels with the statement: "Software developed through Agile methods is relatively less secure when compared to software developed through sequential SDLC processes, like Waterfall". As illustrated in Figure 4.1, participant responses reflected diverse perspectives. The largest segment (43.7%) of respondents disagreed with the statement, suggesting their confidence in Agile methodologies' security capabilities. A significant portion (31%) maintained a neutral stance, neither endorsing

nor rejecting the statement. This neutrality potentially indicated uncertainty regarding Agile practices' security implications. A smaller fraction of participants expressed agreement, with 15.6% strongly agreeing and 9.38% agreeing with the statement. The distribution of responses demonstrated a lack of consensus among participants, highlighting the necessity for deeper investigation into how security practices influence Agile development processes.

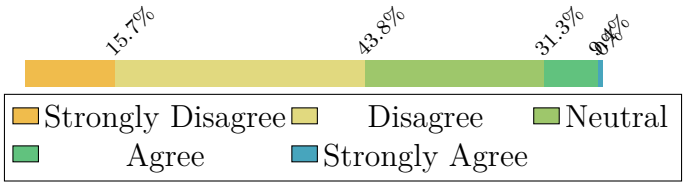


Figure 4.1: Participants' response on, "Agile software is less secure compared to the ones developed using Waterfall".

Effectiveness of software security practices and willingness to include them in Agile

The survey presented the eight state-of-the-art security practices outlined in Table 4.1. Participants evaluated each practice's potential effectiveness for enhancing software security and robustness within their Agile development frameworks. The analysis encompassed responses from 30 participants, with results visualized as a heat map in Table 4.3. The effectiveness ratings revealed distinct patterns across different security activities. "Iterative vulnerability testing" emerged as the second most highly rated practice, with 60% of respondents rating it as "Extremely effective" in vulnerability identification and mitigation. "Automatic testing" received the highest effectiveness rating at 66.67%, highlighting automation's perceived value in strengthening security throughout the development lifecycle. "Iterative security static analysis" and "Additional weight based on security impact" also garnered substantial effectiveness ratings. Conversely, "Iterative risk analysis, countermeasure graphs" received comparatively lower evaluations. Statistical analysis through ANOVA

Table 4.3: Security Activities and Practitioners' Perceived Effectiveness

Security Activity	**Not at all**	**Slightly**	**Moderately**	**Very**	**Extremely**
Addressing security in early iterations with requirements and testing	0%	0%	13.33%	73.33%	13.33%
Stating security requirements that are expected in the production software	0%	3.33%	20%	46.67%	30%
Adding a security specialist to your team	0%	6.67%	20%	40%	33.33%
Additional points or weights to issues with an impact on security	0%	0%	20%	46.67%	33.33%
Iterative and incremental vulnerability and penetration testing	0%	0%	10%	30%	60%
Iterative and incremental security static analysis	0%	3.33%	6.67%	53.33%	36.67%
Iterative and incremental risk analysis, countermeasure graphs	0%	6.67%	30%	43.33%	20%
Automatic testing	0%	3.33%	0%	30%	66.67%

testing revealed no significant variations in perceived effectiveness across the security activities ($F = 1.806, p = 0.09035$). These observations illuminated practitioners' perspectives on security practice effectiveness and emphasized the potential benefits of automated security activities within Agile environments.

The survey examined participants' inclination to incorporate specific security practices from Table 4.1 into their Agile workflows. This section of the study garnered responses from 31 participants. The response distribution, presented in Table 4.4, revealed varying levels of adoption willingness across different practices. Iterative vulnerability testing and iterative security static analysis emerged as highly favored practices. Automatic testing demonstrated the highest acceptance rate, with 67.74% of respondents indicating they were "Extremely willing" to implement this practice. Additional security-weighted prioritization and explicit security requirements documentation also received positive response rates. In contrast, early-

Table 4.4: Security Activities and Practitioners' Willingness to Include Them in Agile Processes

Security Activity	Not at all	Slightly	Moderately	Very	Extremely
Addressing security in early iterations with requirements and testing	0%	0%	29.03%	45.16%	25.81%
Stating security requirements that are expected in the production software	0%	0%	29.03%	41.94%	29.03%
Adding a security specialist to your team	0%	6.45%	19.35%	48.39%	25.81%
Additional points or weights to issues with an impact on security	0%	0%	12.9%	38.71%	48.39%
Iterative and incremental vulnerability and penetration testing	0%	0%	16.13%	19.35%	64.52%
Iterative and incremental security static analysis	0%	0%	3.23%	45.16%	51.61%
Iterative and incremental risk analysis, countermeasure graphs	3.23%	0%	22.58%	48.39%	25.81%
Automatic testing	0%	0%	3.23%	29.03%	67.74%

iteration security implementation and iterative risk analysis garnered lower acceptance levels. Statistical analysis through ANOVA testing indicated no significant variation in adoption willingness across practices ($F = 1.3191, p = 0.2457$). These findings offer guidance for organizations by identifying security practices that Agile practitioners are more willing to adopt and perceive to be more effective, enabling teams to prioritize high-acceptance activities when planning to implement security activities within Agile.

4.3.2 RQ2: Impact on Productivity

Team Velocity

Analysis of responses ($n = 14$) to the optional inquiry regarding security practices' effect on team velocity revealed that most participants observed no significant impact on team

productivity following security measure implementation. The influence varied according to team-specific protocols, illustrated by one participant's observation that activities were structured *"before the sprint starts, [and] the developers go in the sprint knowing what to expect"* (P26). Several participants identified modest productivity variations, noting impacts of *"10-20%"* (P17) or duration extensions of one to two days (P16, P28). A contrasting perspective emerged from one participant who indicated that new security engineering initiatives *"will always affect the sprint velocity drastically"*, resulting in extended feature deployment timelines, though ultimately concluding that *"such changes are fruitful"* (P14).

Day-to-day Activities

The study revealed minimal disruption to daily operations from security measure implementation. A significant proportion of respondents ($n = 16$) reported negligible impact on routine development activities. Non-disruptive practices included the integration of *"periodic security checking"* (P20) tools and external team coordination. The primary operational impact ($n = 2$) centered on extended development timelines, exemplified by P27's observation that security protocols necessitated *"more time spent authenticating to access different environments and projects"*. These findings demonstrate that security practice integration within Agile methodologies maintained overall sprint velocity while introducing minimal operational friction.

4.3.3 RQ3: Impact of Software Activities

Software Products

The investigation examined how security practice integration within Agile methodologies influenced software development practitioners. Analysis of participant responses regarding

security practices' impact on software products revealed that a significant portion ($n = 10$) reported enhanced overall security. This finding suggested that robust security measure implementation positively affected product resilience against potential threats. Additional benefits included increased customer trust ($n = 1$), heightened team confidence ($n = 1$), improved standards compliance ($n = 1$), and reduced bug occurrence ($n = 1$).

The study also identified potential drawbacks of security implementation. One participant noted that these measures *"extended delivery date since a lot of code was often stuck waiting for approval"* (P30). Another respondent (P19) indicated that their teams' security practices *"rarely"* influenced product security. These observations demonstrate that while security practices generally enhanced product security posture, customer trust, team morale, and quality metrics, successful integration required strategic planning to minimize deployment delays.

Organization

Examination of organizational impacts revealed varied effects. Positive outcomes included increased security practice adoption likelihood ($n = 1$), enhanced organizational culture ($n = 1$), strengthened company reputation ($n = 1$), and elevated customer confidence ($n = 1$). However, the majority of participants reported either no effect ($n = 4$) or minimal impact ($n = 4$) on organizational operations. While these findings suggest limited organizational influence, the inherent challenge of quantifying prevented security threats complicated impact assessment without third-party security evaluation.

Confidence

The analysis explored how adopted security practices influenced participants' confidence in their software products' security. Results visualized in Figure 4.2 revealed that 50% of respondents expressed feeling "fairly confident" about their software security, while 25% reported being "somewhat confident," and 25% indicated "complete confidence." These findings demonstrated security practices' positive influence on practitioner confidence levels, though they suggest opportunities for further enhancement of security assurance among software engineers.

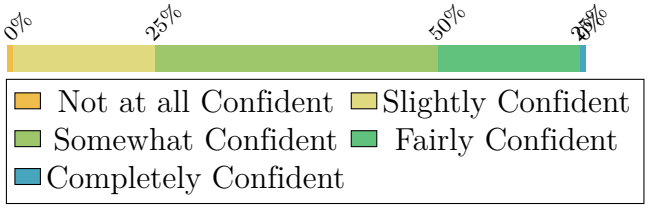


Figure 4.2: Participants' confidence in software security was influenced by the adoption of security practices by their Agile teams

4.4 Discussion

The investigation yielded significant insights into security practice integration within Agile frameworks, illuminating practitioner perspectives and operational impacts across development processes. The analysis revealed predominantly positive perceptions toward security integration within Agile methodologies, despite potential domain conflicts. This receptiveness reflected heightened awareness regarding software security importance. The study demonstrated minimal productivity impact from security integration while generally enhancing software security levels. Although participants noted occasional timeline extensions and feature deployment delays related to security activities, the perceived security benefits out-

weighed these considerations, suggesting that strategic implementation enables successful security integration without compromising productivity.

The research identified opportunities for enhancement, as some participants expressed only "somewhat" or "fairly" confident assessments regarding their adopted security activities' protective capabilities. These observations indicated potential for improvement in Agile security practice effectiveness. To address efficiency concerns and enhance security confidence, we propose focusing on two key areas: *increasing automation* and *improving feedback*.

4.4.1 Increase Automation

The findings highlighted favorable perceptions and adoption willingness toward automated security processes. While statistical analysis revealed no significant variations across the eight examined security activities, automated approaches (e.g., automatic testing) garnered higher acceptance and implementation willingness compared to manual interventions (e.g., security specialist integration). This aligned with previous research demonstrating automated penetration testing's superior efficiency over manual methods [166]. However, Edwards et al. cautioned against potential automated security technique drawbacks, citing social and technical challenges including output inaccuracy, stakeholder requirement misalignment, and information oversaturation [167].

These observations suggested opportunities for enhanced security automation. The results indicate that teams should consider implementing automated testing frameworks, vulnerability assessments, penetration testing protocols, and static security analysis tools, given their higher perceived effectiveness and adoption willingness metrics. Prior research explored continuous security testing integration and automated security tool implementation within CI/CD repository processes [45]. Similarly, DevSecOps emerged as a security-enhanced vari-

ant of DevOps methodology [46], commonly adopted alongside Agile processes due to rapid deployment capabilities [121]. Integration of security activities within these frameworks could further streamline software protection processes.

4.4.2 Improve Feedback

The research cautioned against indiscriminate automation implementation. Previous studies indicated users' reluctance to adopt automated development tools failing to meet personal, interactive, and technical requirements [168]. This study corroborated these findings, with participants reporting minimal perceived security impact and reduced confidence in adopted processes. Edwards et al. noted that contextless information abundance could impede developer tool utilization [167]. Prior research indicated that inadequate feedback inhibited security tool adoption [28, 129]. Enhanced collaboration between security experts and developers [127] could improve automated security tool effectiveness and strengthen engineer confidence.

Research suggested developers' limited understanding of security output implications [28]. Studies explored using vignettes and narratives to enhance security concept comprehension [169]. Additionally, developers demonstrated knowledge gaps regarding security issue remediation. Bhandari et al. addressed this through examination of open-source repository vulnerabilities and corresponding solutions [170]. Research demonstrated automated program repair's effectiveness in debugging tasks [171], suggesting potential benefits from automated security fix suggestions in providing targeted practitioner feedback and enhancing software security.

4.5 Limitations and Future Work

The study's generalizability to all Agile practitioners presented a primary limitation. Although participant recruitment targeted diverse backgrounds and experience levels to comprehensively assess security practices' impact, the substantial representation of graduate students ($n = 11$), predominantly from Virginia Tech ($n = 9$), can potentially introduce sampling bias. Future research could address this limitation through broader practitioner surveys encompassing more diverse populations.

The reliance on participants' retrospective estimations regarding security practices' impact on team velocity can introduce potential accuracy limitations. Additional research examining productivity impact could employ quantitative analysis of Github[12] repositories to evaluate implemented practices and measure objective productivity metrics.

A promising direction for future investigation involves conducting comprehensive case studies examining security practice integration within active Agile development teams. This approach would facilitate collection of direct feedback regarding implementation experiences, challenges, and benefits. Such studies would enhance understanding of team dynamics, workflow impacts, and security measure effectiveness within Agile environments, contributing valuable empirical evidence to existing theoretical frameworks.

Further research opportunities exist in developing innovative security tools and methodologies aligned with Agile principles. Potential areas include: (i) creating documentation reduction systems providing concise & actionable security feedback, (ii) LLMs to synthesize and simplify complex security reports, (iii) developing CI/CD pipeline integration tools for static code vulnerability detection, and (iv) enhancing automation in security practices to streamline integration with existing Agile workflows, particularly within CI/CD and De-

[12] https://github.com/

vOps processes. These developments could address current implementation challenges while maintaining Agile development efficiency.

4.6 Conclusion

This investigation contributed key insights toward enhancing security practice integration within Agile development frameworks. By examining practitioner experiences and perceptions, the study revealed promising evidence supporting the perceived compatibility of security activities with Agile methodologies. The findings demonstrate that security implementation need not compromise development efficiency when properly executed. Specifically, the research identified automation enhancement and feedback optimization as critical success factors - areas directly focused on improving security tool adoption and effectiveness. The study's results regarding practitioners' willingness to adopt automated security activities particularly support the proposed direction for security tool development. The identified challenges in security practice integration provide valuable guidance for future work in developing improved security automation solutions.

Table 4.5: Survey Participants

Participant	**Role**	**Industry Exp. (years)**	**Agile?**	**Security?**
P1	Associate Software Engineer	1	Yes	Yes
P2	Software Engineer	2.2	Yes	Yes
P3	Software Engineer	2	Yes	Yes
P4	Engineering Manager	11	Yes	Yes
P5	Software Engineer	6	Yes	Yes
P6	Student	0	Yes	Yes
P7	Quality Engineer	1.5	Yes	No
P8	Graduate Teaching Assistant	1	Yes	Yes
P9	Student	4	Yes	No
P10	Consultant	15	Yes	Yes
P11	Senior Software Engineer	5	Yes	Yes
P12	Student	3	Yes	Yes
P13	Student	3	Yes	No
P14	Automation Test Engineer	4.2	Yes	Yes
P15	Graduate Student	2.8	Yes	No
P16	Student	0	Yes	No
P17	Senior Software Engineer	6	Yes	No
P18	Chief Test Monkey	41	Yes	Yes
P19	Cloud Engineer	7	Yes	Yes
P20	Systems Architect	8	Yes	No
P21	Department Head	23	Yes	Yes
P22	Associate Director of Systems Development	11	Yes	Yes
P23	Director, DBAA	22	Yes	Yes
P24	Software Developer	20	No	No
P25	Software Engineering Co-Op	1	Yes	Yes
P26	Senior Product Manager	10	Yes	No
P27	Software Engineer	2.5	Yes	Yes
P28	Student	3	Yes	No
P29	Student	1	Yes	No
P30	Technical Consultant	1	Yes	Yes
P31	Software Engineer	2	Yes	Yes
P32	Security Co-Op	0-1	Yes	Yes
P33	Senior Staff Machine Learning Engineer	4	Yes	Yes
P34	Infrastructure Engineer	1.5	Yes	Yes

Chapter 5

Harnessing the Power of LLMs to Simplify Security: LLM Summarization for Human-Centric DAST Reports

5.1 Motivation

DAST tools are used to evaluate the security posture of web applications [18]. These tools analyze applications from an external perspective by simulating real-world attack scenarios, closely resembling the behavior of malicious actors interacting with deployed systems [172]. The result of a DAST scan is typically a detailed report containing numerous security alerts, which are categorized and prioritized based on their assessed severity. Commonly used DAST tools include Burp Suite[1] and ZAP.[2] Such tools play a crucial role in helping software practitioners detect, understand, and remediate vulnerabilities in web applications [173].

However, building and maintaining secure software systems requires substantial expertise across multiple technical domains [174]. Prior research has shown that many software prac-

[1] https://portswigger.net/burp/pro
[2] https://www.zaproxy.org/

titioners struggle to effectively understand and address security issues [22], largely due to limited security knowledge and experience [20]. Interpreting security alerts often demands additional time and cognitive effort, which can be challenging for practitioners already operating under tight development schedules. These challenges are further amplified in fast-paced Agile development environments, where time constraints are more pronounced [21] and developers may exhibit lower willingness to adopt or engage deeply with security-related practices [175].

The ability of LLMs to produce coherent, context-aware, and high-quality textual outputs has attracted considerable attention in recent years [176]. Prior work has demonstrated the effectiveness of LLMs for text summarization tasks, where they can transform lengthy or complex natural language inputs into concise and informative summaries [108, 115, 177]. Motivated by these capabilities, we propose leveraging LLMs to summarize DAST-generated security alerts, with the goal of reducing cognitive overhead and improving the interpretability of security findings for software practitioners. This chapter addresses a critical challenge in security-Agile integration: the complexity and length of DAST tool outputs. This work demonstrates how LLMs can make security practices more accessible in Agile environments by using them to summarize security alerts from popular DAST tools (Burp Suite and ZAP).

Our work explored the following research questions (RQs):

RQ1 What challenges do software practitioners face when dealing with DAST reports?

RQ2 How do software practitioners perceive the effectiveness of LLM-generated summaries of DAST security alerts in understanding the security issue as compared to the original alert?

RQ3 To what extent do software practitioners prefer LLM-generated summaries of DAST security alerts over the original security alerts?

To answer these research questions, we distributed an online questionnaire to 48 professionals working in software development. To gather experimental data, we performed security assessments using two DAST tools—ZAP and Burp Suite—on hackthissite.org,[3] an established platform for security training exercises [178]. For our analysis, we selected one representative security alert from each tool's output. These alerts were given as input to five distinct LLM systems: Orca-Mini,[4] Llama 2,[5] Code Llama,[6] Mistral,[7] and GPT-3.5.[8] Subsequently, we gathered participant feedback on these LLM-processed versions, evaluating their accessibility and understandability.

Our results provide initial insights into how LLMs could transform DAST security alerts into more digestible formats, addressing current limitations in DAST reporting systems. This research represents the first exploration of leveraging LLMs for DAST security alert summarization. Drawing from our findings, we discussed recommendations for developing more user-friendly security alert systems to enhance vulnerability prevention and strengthen software security measures.

5.2 Methodology

5.2.1 Security Report Generation

For our security assessment framework, we implemented a dual-tool approach utilizing Burp Suite (Professional) and ZAP, representing commercial and open-source DAST solutions re-

[3] https://www.hackthissite.org/
[4] Orca-Mini
[5] Llama 2
[6] Code Llama
[7] Mistral
[8] GPT-3.5

spectively. Our security analysis targeted hackthissite.org[9], selected for its open accessibility and extensive array of security vulnerabilities, which provided an ideal testing environment. The security assessment generated two distinct alert sets, from which we extracted one representative alert per tool for our analysis.

The alert summarization phase incorporated five distinct LLM systems. Our selection criteria focused on accessibility through Ollama[10], leading to the inclusion of Orca-Mini, Llama 2, Code Llama, and Mistral, each representing different parameter configurations.

The selected models included several from Meta AI[11]'s Llama architecture family [179]. Among these, Code Llama, an extension of Llama 2, was specifically engineered for code-related tasks [180]. Orca-Mini represented an implementation trained on Orca-style datasets using the Llama and Llama 2 architectures [181]. The Mistral AI[12] 7B parameter model incorporated advanced features like grouped-query and sliding window attention, demonstrating superior performance compared to Llama2 [182]. We also integrated GPT-3.5 through OpenAI's[13] ChatGPT[14] platform, given its widespread adoption and established capabilities. Detailed specifications of these LLMs are presented in Table 5.1.

Table 5.1: LLMs Used in this study

Model	**Parameters (Billions)**	**Size**
Orca-Mini	3B	2 GB
Llama2	7B	3.8 GB
Code Llama	7B	3.8 GB
Mistral	7B	4.1 GB
GPT-3.5	175B	≈700 GB

[9] https://www.hackthissite.org/
[10] https://ollama.com/
[11] https://ai.meta.com/meta-ai/
[12] https://mistral.ai/
[13] https://openai.com/
[14] https://chat.openai.com/

Our methodology involved analyzing actual security vulnerabilities identified in hackthissite.org through ZAP and Burp Suite scans. The detected vulnerabilities were processed through our selected LLM systems using a standardized prompt format:

GIVE SUMMARY FOR: [**Text**]

In this format, "[Text]" represented the core security findings from the original alerts, excluding the Request-Response details. The evaluation process paired original alerts with their corresponding LLM-generated summaries for participant assessment. The security vulnerabilities are documented in Table 5.2.

Table 5.2: Overview of Security Reports

Security Issue	**Traditional**	**LLM**
Vulnerable JavaScript dependency[15]	Burp Suite	Orca-Mini, Llama 2, Code Llama, Mistral, and GPT-3.5
Path Traversal[16]	OWASP ZAP	Orca-Mini, Llama 2, Code Llama, Mistral, and GPT-3.5

The generated summaries by were found to be accurate and reliable. This was supported by the BERT scores (Table 5.3), which demonstrated high accuracy across the models: for ZAP, scores ranged from 0.7913 (Orca-Mini) to 0.8487 (GPT-3.5), and for Burp Suite, scores ranged from 0.8505 (Mistral) to 0.8931 (Llama2). These high scores illustrate the models' ability to produce accurate summaries. In addition, the quality and correctness of the generated summaries were validated through manual human verification.

Table 5.3: BERT scores for summaries generated by different LLMs across DAST tools.

	Orca-Mini	**Llama2**	**Code Llama**	**Mistral**	**GPT-3.5**
ZAP	0.7913	0.8394	0.8309	0.8310	0.8487
BurpSuite	0.8630	0.8931	0.8868	0.8505	0.8697

5.2.2 Participant Recruitment

Our study employed a multi-channel recruitment strategy to assemble a heterogeneous participant sample. The primary recruitment channel leveraged LinkedIn[17], where we distributed both targeted individual invitations and public announcements to attract IT security professionals. To expand our participant base, we activated professional networks to connect with IT personnel across multiple sectors and organizational structures. The participant pool was further diversified by incorporating technically experienced graduate students from Virginia Tech, who brought relevant software development background to the study. This three-pronged recruitment approach enabled us to capture varied professional perspectives and experience levels, enhancing the depth and breadth of our collected data.

5.2.3 Survey Structure

Our survey was developed and distributed using QuestionPro[18], with the survey receiving prior IRB approval. The survey consisted of multiple sections. The initial section captured participants' demographic profiles and assessed their expertise with security reporting systems using a Likert scale from "Very Unfamiliar" to "Very Familiar". We also evaluated participants' confidence levels in interpreting security alerts, with responses ranging from "Very Uncomfortable" to "Very Comfortable". The survey investigated participants' experi-

[17] https://www.linkedin.com/
[18] https://www.questionpro.com/

ence with automated security tools, including usage frequency and practical applications.

To gauge the effectiveness of conventional security reports, participants rated their utility in addressing security concerns on a scale from "Very Useless" to "Very Useful". To address RQ1, we incorporated an open-ended question exploring participants' experiences with traditional DAST report challenges.

For RQ2, the survey presented randomized sequences of original security alerts alongside their summaries for comparative evaluation. Participants assessed each alert's clarity and comprehensibility. The concluding section solicited suggestions for enhancing security report effectiveness through open-ended responses. To address RQ3, we examined participants' preferred formats for receiving security-related information. The complete survey is available in the Appendix B.

5.2.4 Data Analysis

Our research methodology incorporated both quantitative and qualitative data collection through close-ended and open-ended survey questions, examining participants' perspectives on security reporting systems and LLM-generated summaries.

The quantitative analysis employed statistical methods, including Chi-squared (χ^2) tests, to evaluate Likert scale responses regarding clarity and comprehensibility across traditional and LLM-generated reports. For qualitative data, we implemented an iterative-inductive thematic open-coding approach. Two researchers conducted independent analyses, followed by collaborative discussion to achieve consensus on findings.

For example, while addressing RQ1, we analyzed responses from 17 participants who shared their experiences with challenges associated with DAST reports. Via open coding we identified key challenges through sentence-level analysis. For example, in the response *"Lengthy*

security reports can be quite daunting to someone who is seeing them for the first time and does not have the needed level of security knowledge to understand these reports", we identified the *Difficult to understand* challenge. Similarly, from *"Security reports go into very lengthy details which, as a developer, is very hard to follow because I am not a cybersecurity professional"*, we extracted multiple challenges: *Lack of relevance*, *Lengthy*, and *Too detailed*. This systematic analysis revealed recurring patterns in practitioners' experiences with DAST reports. The complete categorization of challenges identified through open coding process is documented in Table 5.4.

5.2.5 Participants

The study garnered responses from 48 software practitioners, collectively representing an average of seven years in software engineering. The participant pool comprised 73% ($n = 35$) industry professionals, averaging eight years of technical expertise, while 27% ($n = 13$) were from the academic sector.

The practitioners worked at diverse organizations including Acquia,[19] GlobalLogic,[20] Palo Alto Networks,[21] and TikTok,[22] spanning roles from product management to systems architecture, software engineering, technical consulting, and engineering management. The academic participants consisted of graduate students who brought an average of two years of software engineering work experience to the study.

As shown in Figure 5.1, there existed varying degrees of security report familiarity among the survey participants, with a significant portion reporting limited exposure: 43.75% ($n = 21$) somewhat unfamiliar and 27.08% ($n = 13$) very unfamiliar. This trend extended to

[19] https://www.acquia.com/
[20] https://www.globallogic.com/
[21] https://www.paloaltonetworks.com/
[22] https://www.tiktok.com/about

Table 5.4: Open coding examples for "What challenges do you face when dealing with traditional DAST security reports?"

Participant	Response	Categories Identified
P1	"The reports are too big and tiresome to read"	Lengthy, Tiring, Overwhelming
P2	"Lengthy security reports can be quite daunting to someone who is seeing them for the first time and does not have the needed level of security knowledge to understand these reports"	Lengthy, Overwhelming, Difficult to understand
P3	"Lengthy security reports are only useful when a software engineer wants to get into detail of the issue and understand it completely. They are cumbersome when we just want to skim over the issue to get a general idea about what the issue is"	Lengthy, Too detailed, Overwhelming, Time-consuming
P6	"Security reports go into very lengthy details which, as a developer, is very hard to follow because I am not a cybersecurity professional"	Lengthy, Too detailed, Lack of relevance
P8	"These reports have details that are of very little use to me"	Unnecessary details, Lack of relevance
P9	"They usually are not solution-oriented"	Not solution-focused
P10	"Traditional, lengthy security reports often provide a detailed analysis of security vulnerabilities, indicating their root causes, possible effects, and methods used. This lengthy report coverage is essential for understanding the complexity of security issues and providing guidance on addressing or fixing the security bug. But one major challenge of traditional, lengthy security reports is that they require a lot of time (time-consuming) to read and understand the report content"	Difficult to understand, Time-consuming
P12	"As mentioned, these reports are quite lengthy, giving too much information"	Lengthy, Too detailed
P13	"While I've never used a security report, I imagine they are lengthy, and it is difficult to understand where to actually make changes to fix the security problems"	Lengthy, Difficult to understand
P17	"In my personal work experience, I have not dealt with security reports apart from the automatic PRs generated by GitHub Dependabot, which are mostly handled by senior engineers of the dev team"	No experience
P18	"Lengthy security reports will contain too much detail for an average software engineer"	Lengthy, Too detailed, Difficult to understand, Lack of Relevance
P19	"Lengthy reports are mostly not for developers who have less security-level expertise"	Lengthy, Difficult to understand, Lack of Relevance
P26	"I would assume they are useful, as the more detailed and verbose the logs are, the better you can understand the security threat"	No experience
P40	"A lot of unnecessary details"	Unnecessary details
P43	"I have not dealt with security reports"	No experience
P46	"I have never used these lengthy security reports, but I have seen them in action, and most developers I know would prefer more automation than having to read lots of information"	No experience, prefer automation
P48	"Typically, I see vulnerability assessments on the libraries and APIs that I use. Sometimes the vulnerability assessments don't give alternatives for the libraries and APIs, which makes it more difficult to steer away from them. They also don't really discuss improper usage and how to avoid improper usage"	Vulnerability assessments, lack of guidance

security issue comprehension confidence, as illustrated in Figure 5.2, where 60.41% ($n = 29$) of respondents indicated being either very uncomfortable or somewhat uncomfortable, suggesting potential accessibility barriers in technical security documentation.

Despite these comprehension challenges, as indicated in Figure 5.3, 83.33% ($n = 40$) of participants had experience with automated security tools in their projects. Their toolkit encompassed various platforms including Burp Suite,[23] ZAP,[24] Acunetix,[25] GitHub Dependabot,[26] and supplementary plugins, though usage frequency varied from monthly to quarterly or less frequent intervals.

The effectiveness of traditional DAST reports received mixed evaluations, as depicted in Figure 5.4. While 31.91% ($n = 15$) maintained a neutral stance, a substantial 48.94% ($n = 23$) rated them as somewhat or very useless. Only 19.15% ($n = 9$) found these reports somewhat or very useful, highlighting opportunities for enhancing report accessibility and practical value. Comprehensive participant demographics were documented in Table 5.5.

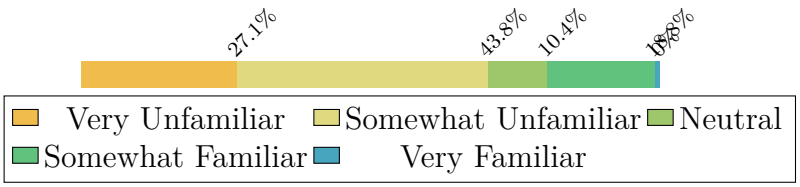


Figure 5.1: How familiar are you with security reports of software products?

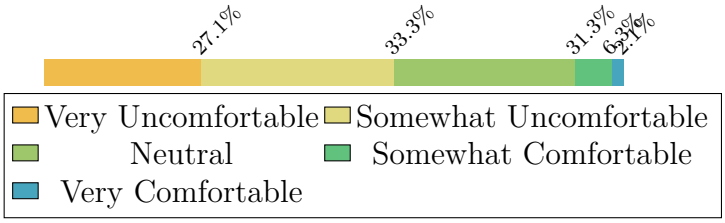


Figure 5.2: How comfortable are you with knowing and understanding security issues mentioned in security reports?

[23] https://portswigger.net/
[24] https://www.zaproxy.org/
[25] https://www.acunetix.com/
[26] https://github.com/dependabot

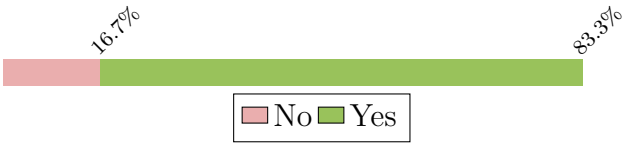


Figure 5.3: Have you used automated security tools to scan and identify software security issues?

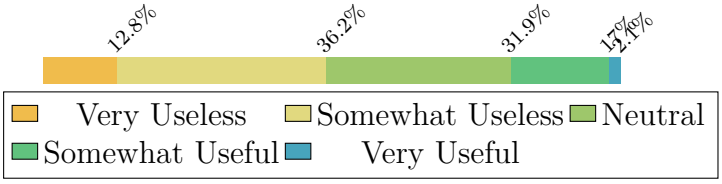


Figure 5.4: How would you rate the usefulness of traditional DAST reports in addressing and resolving security issues?

5.3 Results

5.3.1 RQ1: Challenges with traditional DAST reports

Through thematic open coding analysis of survey responses (Table 5.4), we observed several challenges emerged in practitioners' interactions with DAST reports. The analysis revealed six major challenge categories: **Difficult to Understand** ($n = 5$), **Too Detailed** ($n = 2$), **Lack of Relevance** ($n = 4$), **Lengthy** ($n = 8$), **Overwhelming** ($n = 3$), and **Time Consuming** ($n = 3$).

These identified challenges presented substantial impediments to practitioners' operational efficiency and security management capabilities. The complexity and volume of traditional DAST reports potentially compromised organizational security by creating barriers against a thorough review processes. The combination of comprehension challenges, contextual irrelevance, and time-consuming analysis requirements results in decreased engagement with security documentation, potentially leaving critical vulnerabilities unaddressed [183]. The cumulative impact of these challenges in DAST reports suggests a need for more streamlined,

Table 5.5: Survey Participants

Participant	Role	Industry Exp. (years)	Participant	Role	Industry Exp. (years)
P1	Engineering Manager	12	P2	Staff Software Engineer	9
P3	Software Engineer	3	P4	Associate Engineer	2.5
P5	Assistant Manager	4	P6	Senior Staff ML Engineer	4
P7	Student	3	P8	Software Engineer	4
P9	Graduate Research Assistant	8	P10	Student	0
P11	Software Engineer	3	P12	Senior Software Engineer	7
P13	Student	0	P14	Student	0
P15	Student	0	P16	Software Engineer	2
P17	Software Engineer	3.5	P18	Software Engineer	3
P19	Software Engineer	4	P20	Director of Software Engineering	12
P21	Software Engineer	8	P22	Software engineer	3
P23	Student	5	P24	Consultant	3
P25	Senior Software Engineer	3	P26	Student	0
P27	Student	2	P28	Consultant	20
P29	Senior Software Engineer	6	P30	Student	5
P31	Quality Engineer	5	P32	Software Engineer	2.5
P33	Associate Specialist SAP	4	P34	Senior Product Manager	11
P35	Associate Software Engineer	3.5	P36	Director	22
P37	Associate Director	13	P38	Department Head	25
P39	Systems Architect	25	P40	Test Engineering Manager	41
P41	Senior Software Engineer	9	P42	Staff Software Engineer	13
P43	Software Engineer	3	P44	AI Architect	2.5
P45	Developer	3.5	P46	Student	1
P47	Student	0.5	P48	Teaching Assistant	1

accessible reporting mechanisms that could better serve practitioners' security assessment needs while maintaining comprehensive coverage of security concerns.

5.3.2 RQ2: Effectiveness of Security Alerts vs their LLM-generated Summaries

Our evaluation framework for assessing LLM-generated summaries focused on two fundamental performance metrics:

- **Clarity:** The assessment employed a graduated scale from "Very unclear" to "Very clear", examining multiple textual elements including structural organization, grammatical precision, formatting consistency, and overall content quality. This comprehensive evaluation helped determine the effectiveness of information presentation in the summaries.
- **Comprehensibility:** The study implemented a binary evaluation mechanism ("Yes"/"No") to measure participants' ability to grasp the security concerns detailed in each summary. This direct measurement approach aimed to evaluate the summaries' success in conveying critical security information.

Clarity

As shown in Table 5.6, the analysis of survey responses revealed a significant advantage of LLM-generated summaries over original alerts. In the evaluation of Burp Suite alerts, the original alerts presented significant clarity challenges, with 70.83% ($n = 34$) of participants rating it as very unclear. In contrast, the LLM-generated versions demonstrated superior

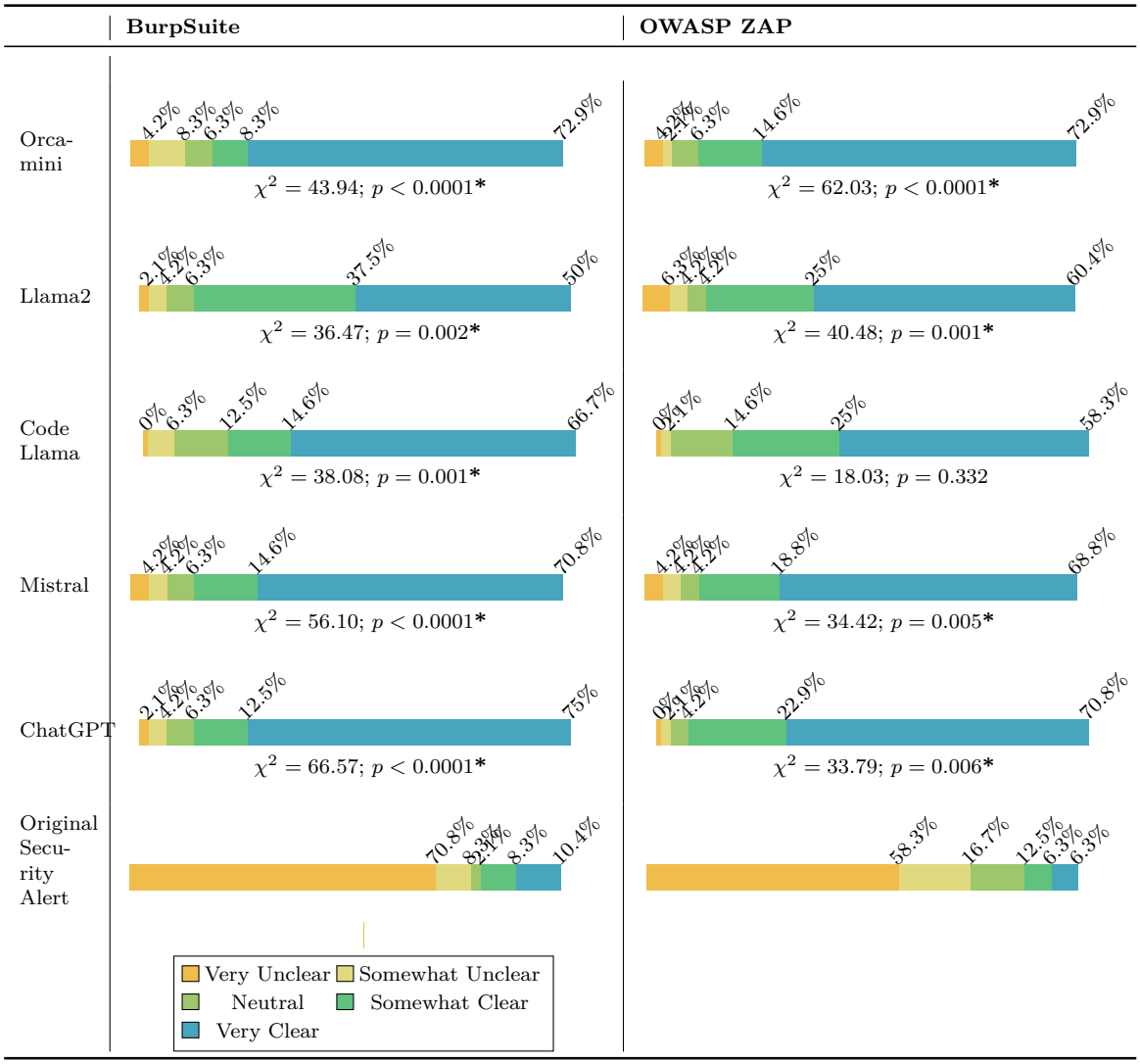
clarity metrics. GPT-3.5 emerged as the leading performer, achieving a 75.00% ($n = 36$) very clear rating. Close behind were the Orca-Mini and Mistral implementations, garnering very clear ratings from 72.92% ($n = 35$) and 70.83% ($n = 34$) of participants respectively. Code Llama and Llama2 also demonstrated improved clarity compared to the original format, securing very clear ratings from 66.67% ($n = 32$) and 50% ($n = 24$) of participants respectively.

The ZAP alert analysis revealed similar patterns, with 58.33% ($n = 28$) of participants rating the original alerts as very unclear. The LLM-processed versions again demonstrated enhanced clarity metrics. GPT-3.5 maintained its leading position with 70.83% ($n = 34$) very clear ratings, followed by strong performances from Orca-Mini at 72.92% ($n = 35$) and Mistral at 68.75% ($n = 33$). Code Llama and Llama2 summaries also received positive evaluations, achieving very clear ratings from 58.33% ($n = 28$) and 60.42% ($n = 29$) of participants respectively. Statistical analysis confirmed significant clarity improvements across all LLM-generated versions compared to traditional reports, with the sole exception of Code Llama's processing of the ZAP path traversal vulnerability.

Comprehensibility

Building on the clarity assessment, our comprehension analysis (Table 5.7) yielded compelling evidence regarding participants' ability to grasp security concepts across different DAST alert presentations. The evaluation of Burp Suite alerts revealed significant comprehension challenges with the original alerts, as 77.08% ($n = 37$) of participants reported difficulty understanding the content. In contrast, LLM-processed versions demonstrated significant improvements in accessibility. GPT-3.5 achieved exceptional comprehension rates, with 93.75% ($n = 45$) of participants successfully understanding the security implications. Mistral and Llama2 implementations also showed substantial gains, achieving comprehen-

Table 5.6: Clarity of Security Alerts compared with their LLM-generated summaries



* denotes statistically significant results (**p-value** < **0.05**)

sion rates of 91.67% ($n = 44$) and 89.58% ($n = 43$), respectively. The Orca-Mini and Code Llama versions maintained strong performance levels, with 87.50% ($n = 42$) and 83.33% ($n = 40$) of participants indicating successful comprehension.

The ZAP assessment revealed similar patterns, with 87.50% ($n = 42$) of participants reporting comprehension difficulties with the original alerts. The LLM-processed versions demonstrated significant improvements in understanding. GPT-3.5 and Code Llama emerged as the

most effective models, both achieving 95.83% ($n = 46$) comprehension rates. Orca-Mini and Llama2 maintained strong performance levels with 89.58% ($n = 43$) and 85.42% ($n = 41$) comprehension rates respectively, while Mistral achieved an 83.33% ($n = 40$) success rate.

Statistical analysis confirmed significant improvements in comprehension across all LLM-processed versions compared to traditional reporting formats. These findings, demonstrate the enhanced effectiveness of LLM-generated summaries in conveying security-critical information. The results established that software practitioners found LLM-processed security alerts substantially more accessible and comprehensible compared to traditional DAST report formats.

5.3.3 RQ3: Preference

Our analysis of preferred security alert formats revealed a decisive inclination towards LLM-summarized versions. As illustrated in Figure 5.5, a substantial majority (81.25%, $n = 39$) of participants expressed a preference for summarized formats. The remaining responses were distributed between those expressing no specific preference (6.25%, $n = 3$) and those falling into alternative categories (8.33%, $n = 4$).

The "Other" category encompassed diverse response patterns. Some participants indicated neutrality with responses such as "*no preference*", while others proposed hybrid approaches that defied strict categorization. For instance, one response suggested "*Unsure specifics, but probably one with a mixture of easy-to-read the summary and necessary technology codes*".

The survey responses illuminated key preferences in security report delivery mechanisms. While summarized formats dominated participant preferences, several respondents advocated for a hybrid approach that balances concise summaries with access to comprehensive technical details when required. Some participants specifically highlighted the value

Table 5.7: Understanding of Security Issues mentioned in Reports

* denotes statistically significant results (**p-value** $<$ **0.05**)

of receiving updates through email or collaborative platforms such as Slack.[27] These findings indicated that while LLM-generated summaries represented the preferred format for security alerts, practitioners valued the flexibility to access detailed documentation when circumstances demanded deeper technical analysis.

[27] https://slack.com/

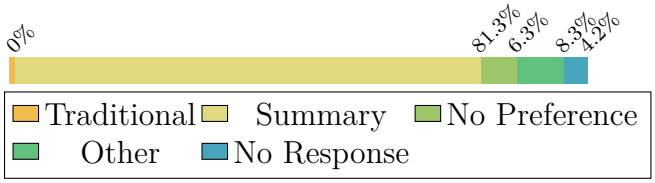


Figure 5.5: If given a choice, which format would you prefer for receiving information about security issues in software projects?

5.4 Discussion

Our investigation uncovered several critical obstacles practitioners encountered with DAST reports, including information overload, contextual misalignment, excessive length, and resource-intensive analysis requirements. These challenges potentially impact operational efficiency and security risk management. We saw that LLM-generated summaries significantly enhanced security alert accessibility and comprehension through streamlined information presentation. Participant responses indicated strong preference for LLM-summarized formats, citing improved clarity and comprehensibility compared to traditional alert mechanisms. In Chapter 4 we saw that there exists resistance to security practice adoption among Agile teams. LLM-generated security alert summaries could facilitate DAST adoption by improving the understanding of security related documentation within Agile's rapid iterations. Based on these findings, we developed guidelines for human-centric security reporting.

5.4.1 Guideline: Concise Reports

Statistical analysis revealed significantly higher clarity and comprehension ratings for LLM-generated summaries compared to traditional DAST reports. GPT-3.5 demonstrated particular effectiveness in alert summarization, achieving superior clarity and comprehension metrics across both testing platforms. Additional models, including Mistral, Llama2, Code

Llama, and Orca-Mini, also showed significant improvements in security issue comprehension.

The strong preference for summarized reports (81.25%, $n = 39$) aligned with previous research indicating practitioner preferences for concise communication in various contexts, including email correspondence [184], code review feedback [185], and automated recommendations [186]. Given LLMs' demonstrated effectiveness in natural language [177] and code [187] summarization, we propose integrating generative AI capabilities into DAST tools for enhanced vulnerability communication.

5.4.2 Guideline: Customized Reports

While quantitative data showed strong preference for LLM-generated summaries (Figure 5.5), qualitative responses highlighted continued value in traditional reporting formats. Some participants advocated for hybrid approaches combining condensed summaries with detailed technical documentation. Several practitioners emphasized the importance of integration with established communication platforms like email and Slack for workflow optimization.

These insights suggested implementing customizable reporting features in DAST tools, accommodating diverse practitioner needs from high-level overviews to detailed analysis. Previous studies identified comprehension barriers in static analysis tool adoption [188] and security vulnerability remediation [28]. Given LLMs' demonstrated success in providing customized solutions across various domains including education [189], healthcare [190], and robotics [191], we propose leveraging LLM capabilities for personalized DAST report generation. This approach could enhance vulnerability mitigation efficiency through customized messaging across different communication platforms, accommodating diverse professional roles and expertise levels.

5.5 Limitations

External Validity While our study yielded promising insights, some constraints warrant consideration. The generalizability of our findings faced potential limitations due to sample size constraints. Although we implemented diversity-focused recruitment strategies to capture varied professional perspectives, the inclusion of 27% ($n = 13$) graduate students from Virginia Tech presented a possible source of sampling bias. A more extensive participant pool would have enabled more comprehensive evaluation of LLM-summary effectiveness in security alert contexts.

Our investigation's scope encompassed two specific DAST tools: Burp Suite and ZAP. Despite their widespread adoption, this selective focus potentially limited broader applicability across diverse security tool-sets and automated testing frameworks. Future investigations could benefit from expanding the tool coverage to evaluate LLM-summary performance across varied security platforms. Furthermore, our exclusive focus on DAST tools potentially limited the applicability of findings to other security testing paradigms, such as SAST implementations.

The study's reliance on five specific LLM implementations, while yielding effective results, potentially overlooked capabilities offered by alternative or newer model architectures. Additional research exploring diverse model implementations could reveal enhanced summarization capabilities and relevance metrics.

Internal Validity The research design presented several methodological constraints. While our comparative analysis examined clarity differentials between traditional and LLM-generated DAST report summaries, it did not incorporate accuracy assessments of the summarized content. The study's emphasis on practitioner perceptions necessitated future investigation into

LLM-generated summary accuracy metrics. Additionally, the thematic analysis methodology applied to open-ended responses introduced potential interpretative bias.

Our methodology's reliance on online survey data collection potentially limited deeper contextual insights. A more comprehensive research approach incorporating qualitative interviews and observational studies within established DAST-integrated teams could have provided richer insights into real-world report utilization patterns and revealed nuanced operational challenges and benefits.

5.6 Future Work

Our research findings suggest several promising directions for future investigation into LLM applications within security vulnerability reporting frameworks. These potential research trajectories focus on enhancing alert accessibility and usability across diverse stakeholder groups in software development environments.

One significant research opportunity lay in the integration of automated security tools and LLM systems within CI/CD pipelines. This integration could facilitate automated security scanning during pull request (PR) submissions. The proposed workflow would generate comprehensive security reports, process them through LLM summarization, and automatically post condensed findings as PR review comments. This streamlined approach could enhance developer efficiency in identifying and addressing security vulnerabilities.

Another research direction warranted exploration of interface design optimization for summary presentation. This investigation could focus on developing intuitive visualization formats that prioritize information accessibility. The research could examine role-specific summary customization—creating distinct formats for managers, developers, and testing

teams—to optimize information relevance and stakeholder engagement. Additionally, investigating LLM summary integration with collaborative platforms could enhance real-time communication and decision-making processes while accommodating diverse accessibility requirements.

Further research opportunities existed in the development and optimization of specialized security-focused LLMs. This approach could incorporate user feedback mechanisms within generated summaries, enabling quality assessment and relevance tracking. The accumulated feedback data could drive continuous model improvement, resulting in increasingly precise and stakeholder-aligned summary generation. This iterative refinement process could potentially lead to more efficient security issue identification and resolution protocols, ultimately enhancing overall project security standards.

5.7 Conclusion

Our research findings provide substantial empirical evidence supporting the integration of DAST security practice into Agile development through LLM enhancement. By investigating practitioner experiences with DAST reports, we uncovered critical barriers to security adoption, including comprehension difficulties, information overload, and time constraints—challenges particularly relevant to Agile's rapid development cycles. Through empirical assessment of 48 software practitioners' responses, we demonstrated that LLM-generated summaries significantly improved alert clarity and comprehensibility, with 81.25% ($n = 39$) of participants preferring summarized formats. This strong preference indicates that LLM-generated security feedback inclusion could help better integration of DAST in Agile workflows. The proposed evidence-based implementation guidelines for human-centric security reporting align with Agile principles of efficiency and iterative improvement, while the

demonstrated effectiveness of LLM-summarized reporting supports flexible security integration across different team roles. The chapter contributes to the dissertation's thesis by empirically documenting developer challenges, demonstrating LLM enhancement as an effective solution, providing implementation strategies, and establishing a framework for security practice integration—such as DAST—within Agile workflow constraints. These findings provide a strong foundation for our future proposed work in developing an automated integration systems to provide LLM-generated security feedback on DAST alerts in the CI/CD pipeline.

Chapter 6

Security Practices in Agile: A Real-World Case Study on Integrating DAST

6.1 Motivation

Modern organizations increasingly rely on *web applications*—software systems accessed through web browsers—to deliver critical services to users across a wide range of domains [192, 193]. To support the rapid evolution of these applications, development teams commonly adopt Agile software development methodologies, which have become the dominant development paradigm in industry [33]. Reports indicate that over 70% of organizations in the United States use Agile approaches [1], and these methods are frequently applied in the development and maintenance of web applications [16, 194]. In parallel, CI/CD practices are widely employed to automate building, testing, and deploying code changes [195, 196], with empirical studies showing strong adoption of CI/CD in web application development contexts [197].

Web application security has become an increasingly critical concern. A substantial proportion of real-world security breaches can be exploited through vulnerabilities in web applications [198, 199], and the growing scale and sophistication of online threats continue to

exacerbate these risks [200]. This challenge is particularly acute for applications that process sensitive user information or must comply with strict data protection requirements [10, 11]. As web applications increasingly collect and manage user data [17], ensuring their security throughout the software lifecycle is essential. However, prior research suggests that integrating security into Agile workflows remains difficult [7].

DAST has emerged as a widely used practice for automating security testing of web applications [18]. DAST tools evaluate applications from an external perspective by dynamically simulating attacks that resemble the behavior of malicious adversaries [172]. The resulting security reports typically rank discovered vulnerabilities by severity, enabling teams to prioritize remediation efforts and address high-risk issues before deployment [173]. Despite the growing availability and adoption of DAST tools, their effective integration into Agile and CI/CD workflows remains under-explored in empirical research, especially in environments where development speed may appear to conflict with security objectives. In particular, little is known about how practitioners perceive DAST in real-world settings, how it affects day-to-day development work, and what challenges arise when balancing security requirements with development velocity.

In this chapter we examined the integration of DAST into Agile workflows in an industrial context. Through an in-depth action research case study, complemented by qualitative interviews with team members, we investigated how DAST was introduced, operationalized, and experienced within a Kanban-based development process. By grounding the analysis in real development practices, this study provides concrete evidence and actionable guidance for incorporating security activities into Agile processes. Specifically, we examine both the perceived benefits and the challenges associated with deploying DAST mechanisms in Agile environments. The study is guided by the below research questions:

RQ1 To what extent would the members of a Kanban-Agile team be willing to take-on and sustain the use of DAST within their development activities, and which factors shape this willingness?

RQ2 What challenges arise during the integration of DAST into Kanban workflows, and how do practitioners perceive its impact on their everyday development work?

RQ3 What changes or enhancements can be introduced to more effectively embed security practices within Kanban-based development processes?

RQ4 How do practitioners perceive using DAST in comparison to other security practices they have previously used or are familiar with?

We addressed these research questions through an action research case study [201] conducted with a real-world software development team that initiated the adoption of DAST within contemporary development practices, specifically Kanban and CI/CD. The author/researcher was embedded within the development team and played a leading role in designing and integrating DAST scans into the CI/CD workflow. The study focuses on the technical, procedural, and human aspects of DAST integration in Kanban and CI/CD contexts, capturing the perspectives of multiple stakeholder roles, including developers, testers, data analysts, and managers. To obtain in-depth insights, we conducted qualitative interviews with ten team members, enabling a detailed examination of both the challenges encountered and the perceived impacts of introducing security practices into modern Agile workflows.

The study makes three primary contributions:

1. Offers a detailed real-world case study of integrating DAST into a Kanban-based web development team, highlighting both technical and organizational considerations.

2. Presents empirical evidence derived from qualitative interviews with practitioners across diverse roles, shedding light on factors that influence the adoption and sustained use of DAST in Kanban and Agile settings.

3. Distills practical recommendations and lessons learned that can inform industry teams seeking to improve the integration of DAST and related security practices into their development workflows.

Collectively, these contributions advance the understanding of how security activities such as DAST, can be effectively embedded within modern Agile development processes.

6.2 Case Study Setting

The case study was carried out within an Identity Services group that operates as part of a broader Information Technology (IT) organization. To protect confidentiality and prevent the exposure of sensitive security information, the organization is described anonymously throughout this dissertation. The Identity Services group was tasked with overseeing the secure handling of digital identities across the organization. Its responsibilities included delivering authentication and authorization mechanisms that regulated access to applications, systems, and data resources. These services supported essential organizational activities, including operational processes, research initiatives, and external engagement. The team's mission centered on delivering reliable identity and access services while continuously improving security posture, usability, and adaptability in response to evolving industry standards and threats.

6.2.1 Team Structure

The team comprised of 23 individuals and was organized into three primary units: Developers ans DevOps engineers, Testers, and Identity & Access Management (IAM) Analysts. Together, these groups worked closely together to design, implement, and maintain secure identity-related services within an Agile development environment. The team was responsible for the operation and continuous improvement of several core systems, including a web-based Single Sign-On (SSO) service, an account provisioning platform, a directory administration application, and an enterprise directory used for IAM. Each functional unit contributed specialized expertise throughout the software development life-cycle: Developers and DevOps engineers focused on implementing new features, maintaining existing ones and managing infrastructure, Testers were responsible for validating system functionality & quality, and Analysts provided requirements-driven insights and assess system behavior and performance. All team members worked remotely and were based within the same time zone.

6.2.2 Development Process

The team employed the Kanban(Agile) development approach, which emphasizes a steady flow of work and the use of work-in-progress limits to enhance productivity and improve the predictability of software delivery. The members of the team had previous experience with Agile and Kanban. Stand-up meetings were held each morning to share progress updates, identify blockers, and outline immediate plans. These meetings were often followed by optional, focused discussions—referred to as after-Kanban conversations—where specific technical or process-related issues could be explored in greater depth. Work was tracked using three separate Kanban boards, one for each subgroup. The combination of regu-

lar stand-up meetings and flexible follow-up discussions promoted both accountability and adaptability, creating a development environment well suited for the introduction of additional security practices [38], such as DAST. For source code management and automation, the team relied on GitLab,[1] which was used both for version control and for configuring and executing CI/CD pipelines.

6.3 Methodology

We employed an action research case study approach [201, 202] conducted in collaboration with an industrial software development team, to gain a deep understanding of practitioners' experiences related to incorporating DAST tools into Agile development workflows. Case studies are widely regarded as an effective research method in software engineering, as they enable the investigation of phenomena within authentic industrial settings [203, 204]. Consistent with prior work that has examined software engineering practices in similar situations [205, 206, 207], our study adopted this approach to capture both technical and organizational dimensions of DAST adoption. The research followed the five iterative phases of the action research cycle [201, 202] described below.

6.3.1 Problem Diagnosis

The problem diagnosis phase aims to identify and characterize challenges as they manifest in industrial practice [201]. Prior research has indicated that software teams following Agile processes often encounter difficulties when attempting to incorporate security-related activities into their development workflows [175]. Moreover, user interfaces—particularly those of web applications—are known to be complex to design [208], test [209], and secure [210]. A

[1] https://about.gitlab.com/

wide range of security vulnerabilities, including Cross-Site Scripting,[2] Information Leakage,[3] Broken Access Control,[4] etc., can be exploited through web-based user interfaces [211]. To address such threats, DAST evaluate web-application security by emulating attacks in the applications external interfaces [18]. Despite their availability, empirical evidence on how DAST is adopted and used in industrial Agile settings remains limited.

Within the case study, the primary objective was to investigate feasible ways of incorporating DAST into Agile. At the outset, the team had minimal security tests in place and largely depended on manual and ad hoc checks to assess system robustness. This limitation motivated the exploration of DAST as a potential solution. An initial task related to DAST integration was added in *Backlog* column of the Kanban board. This task was subsequently picked by an engineer from the testing unit, making this the initiation of the team's effort to adopt DAST. As the task progressed to the *Implement* column, it became the central theme of our research. The initiative to integrate DAST was assigned to the testing unit, which included the embedded author/researcher. They expressed interest in extending their existing test suites to include security-focused tests, recognizing that their current practices did not address security concerns. This recognition prompted an investigation into how DAST could be systematically incorporated into their established Agile workflows.

The scope of this case study examines how a real-world software development team (Section 6.2) adopts DAST within Kanban and CI/CD processes to enhance the security of web applications. To study this process, we employed focused observation [212], a field research technique in which observers concentrate on specific phenomena as they occur naturally. This method was applied with a high level of researcher participation and a low degree of observer obtrusiveness, in line with established guidelines for industrial case studies [203].

[2] Cross-Site Scripting
[3] Information Leakage
[4] Broken Access Control

The author/researcher actively participated as a team member during the DAST integration effort, collaborating closely with the testing group, the full team of 23 members, and a manager to support the incorporation of security testing into the project workflow. Over a period of three months, the author/researcher engaged in routine team activities, including daily stand-ups and after-Kanban discussions. Observations were conducted synchronously through in-person engagement, video conferencing tools (e.g., Microsoft Teams and Zoom), and communication platforms such as Slack. All observations were aligned with routine team practices and activities to reduce disruption and ensure that work proceeded as naturally as possible. Participant anonymity and confidentiality were maintained throughout the study, and the research activities were conducted with the approval of the team's management.

6.3.2 First Iteration

Action Planning

Action planning focuses on identifying and evaluating potential approaches for addressing the diagnosed problem [201]. In this study, the plan was collaboratively developed by the author/researcher and the development team, taking into account the organizational context, including existing infrastructure and available resources. The team articulated a set of functional requirements for automating DAST within their development workflow:

1. The system should be able to authenticate as an authorized user.

2. Once authenticated, execution control should be transferred to a DAST tool.

3. Operating with authenticated privileges, the system should perform security scans to identify vulnerabilities and verify compliance requirements.

Meeting these requirements necessitated the selection of a DAST solution that could be invoked and controlled programmatically, enabling seamless integration into the existing CI/CD pipeline. Based on this criterion, we evaluated eight candidate DAST tools and ultimately selected ZAP[5] as the most appropriate option. ZAP is an open-source, free, cross-platform DAST tool [105]. It offers automated vulnerability scanning capabilities as well as an interactive environment for manual security testing. Crucially, ZAP exposes a local API that enables external control via code, making it well suited for integration into the team's automated CI/CD workflow.

Action Taking

The action taking phase involves executing the planned intervention [201]. In this iteration, the intervention consisted of integrating the selected DAST tool, ZAP, into the team's Agile development process to enhance application security. The researcher introduced this activity as a *direct intervention* [202], as it resulted in concrete changes to the team's operational practices. Next the author/researcher implemented a Java-based program that leveraged existing in-house software to authenticate to the target web application using valid user credentials. After successful authentication, the program extracted the required session artifacts, including authentication tokens and cookies, and transferred them to the ZAP execution environment. This enabled ZAP to interact with the application as an authenticated user, ensuring that security scans were performed with appropriate access privileges. Subsequently, the Java program communicated with a locally deployed ZAP instance via its API to configure and initiate the desired scans. Upon completion of the scans, the program generated a security report that summarized detected vulnerabilities and provided actionable information for the team. This approach enabled the testing group to incorporate DAST

[5] https://www.zaproxy.org/

scanning into the CI/CD pipeline while preserving existing development practices, thereby embedding security checks throughout the software development life-cycle. Following the successful development of the initial Java-based integration, the team evaluated the solution across multiple web applications. While the approach proved effective for relatively simple applications built primarily with HTML and CSS, limitations emerged when applying the solution to more complex, JavaScript-intensive systems. In particular, ZAP exhibited difficulties in accurately processing highly dynamic content generated by modern JavaScript frameworks, reducing scan coverage and effectiveness.

1. **Dynamic Content:** ZAP had limited capability to fully interact with and analyze dynamically rendered elements, which constrained its effectiveness for applications with JavaScript rich client-side behavior.

2. **HTTP Protocol:** In addition, at the time of deployment, ZAP provided support for the HTTP/2 protocol, whereas several of the team's applications had transitioned to the newer HTTP/3 standard. This protocol mismatch further restricted ZAP's applicability within the team's testing environment and motivated the exploration of alternative DAST solutions.

6.3.3 Second Iteration

Action Planning

Following the limitations identified in the first iteration, the team evaluated several alternative DAST solutions, including WebInspect[6] and Burp Suite.[7] Based on the available feature sets and feedback from team members, the decision was made to proceed with Burp Suite.

[6] WebInspect
[7] https://portswigger.net/burp/pro

Burp Suite is a commercial web application security testing platform developed by PortSwigger.[8] Similar to ZAP, it supports both automated vulnerability scanning and manual security testing. In addition, Burp Suite provides a rich ecosystem of built-in extensions—such as Proxy, Intruder, and Repeater—that are commonly used to support penetration testing and in-depth security analysis [106]. These capabilities aligned well with the team's need to assess complex, modern web applications.

Action Taking

As an initial step, the team experimented with the community (free) version of Burp Suite to evaluate its suitability for performing authenticated security scans. With guidance from Burp Suite's support personnel, the team obtained the necessary scripts and configuration settings required to execute scans while authenticated as a valid user. This setup enabled the team to achieve the desired scanning behavior and validated Burp Suite's applicability to their use case. Adopting Burp Suite addressed several shortcomings encountered with ZAP, particularly in handling JavaScript-intensive applications and resolving protocol compatibility issues. This transition represented a critical milestone in ensuring that the DAST solution was both technically effective and compatible with the team's web application stack. After confirming that Burp Suite satisfied the functional requirements, the team acquired a Burp Suite Professional subscription to access advanced features and dedicated support. Despite these improvements, subsequent after-Kanban discussions revealed additional challenges following the integration of DAST.

1. **Timing:** During one such discussion, the team agreed to schedule DAST scans on a quarterly basis in an effort to balance security considerations with development timelines. However, this approach stands in tension with Agile and CI/CD principles,

[8] https://portswigger.net/

which emphasize the frequent execution of automated tests to continuously assess and maintain software quality [196].

2. **Lack of Bandwidth:** Team members also reported limited capacity to triage and remediate the full volume of security alerts produced by DAST scans. As a result, the team collectively decided to prioritize the resolution of medium- and high-severity findings. This prioritization strategy allowed critical vulnerabilities to be addressed while avoiding excessive disruption to ongoing development activities.

6.3.4 Evaluation

The evaluation phase aims to examine and interpret the outcomes of the actions undertaken during the earlier iterations [201]. To assess the effects of integrating DAST into the team's Agile workflow, we conducted qualitative interviews with practitioners involved in the project. The interview protocol and study procedures were reviewed and approved by the IRB.

Data Collection

Participant Recruitment Following the completion of the DAST integration, we initiated a series of qualitative interviews to better understand how the security practice was adopted and perceived within the team's Agile processes. Participation requests were distributed through the team's dedicated Slack channel, inviting all the team members to participate. Out of the 23 team members contacted, 10 agreed to participate, resulting in a response rate of 43%. Individual follow-ups were conducted to schedule virtual interview sessions at times convenient for each participant, thereby minimizing disruption to their regular work and ongoing responsibilities. As summarized in Table 6.1, the participant pool

Table 6.1: Interview Participants

Group	Participant	Role	Industry Exp.(Years)
Testing	T1	Test Engineer	20
	T2	Test Engineering Manager	41
	T3	Test Engineer	37
Developer & DevOps	D1	Applications Developer	22
	D2	Middleware Engineering Manager	25
	D3	Lead Software Developer	17
	D4	Database Administrator	40
IAM Analysts	I1	IAM Business Analyst	19
	I2	IAM Systems and Directory Administrator	26
	I3	IAM Systems Analyst	37

To indicate participants' roles, we use the prefix `T-` for testers, `D-` for developers, and `I-` for IAM analysts.

included four members from the Developer and DevOps unit, three from the Testing and the IAM Analyst unit each. This relatively balanced representation across roles enabled the collection of diverse perspectives on the DAST integration effort and its impact on different responsibilities within the team.

Interview Design We designed the interview questions (presented in Table 6.2) to align directly with the research questions and to capture multiple dimensions of the team's experience with integrating DAST into a Kanban-based development process. All questions were open-ended, allowing participants to elaborate on their experiences and perspectives. Interviews were conducted remotely and recorded using either Microsoft Teams or Zoom, depending on participant preference.

To address RQ1, questions Q1 and Q2 focused on participants' initial willingness to adopt DAST as well as their inclination to continue using it over time. These questions helped

Table 6.2: Interview Questions

RQs	No.	Questions
RQ1	Q1	"How willing were you to include the DAST (Dynamic Application Security Testing) security scanning practice into the software development process?"
	Q2	"Now that it has been brought into the Agile process, how willing are you to continue DAST security scanning practice?"
RQ2	Q3	"What impact did the inclusion of this security practice have on your day-to-day work?"
	Q4	"What are the general challenges you had while incorporating security practice into your agile process?"
	Q5	"How did this inclusion affect the team velocity?"
	Q6	"Do you think the software product is more or less secure now that we have included this security practice?"
RQ3	Q7	"What improvements can be made to better integrate security practices into Agile?"
RQ4	Q8	"How does the use of DAST compare to other security practices familiar to team members in Agile?"

uncover factors influencing openness to incorporating security testing into daily development activities and Agile routines.

Q3, Q4, Q5 & Q5 addressed RQ2 by examining the practical effects of DAST on the team's workflow. Specifically, Q3 and Q4 surveyed the challenges and disruptions encountered during day-to-day work, including how DAST aligned with existing Agile practices. Q5 probed participants' perceptions of DAST's influence on development velocity, while Q6 captured broader views on whether the overall security posture of the software improved after DAST was introduced.

Finally, questions Q7 and Q8 were used to address RQ3 and RQ4. These questions elicited participants' suggestions for improving the integration of security practices within Agile workflows and their comparisons of DAST with other security techniques they had previously encountered. Together, these responses helped situate DAST within the broader ecosystem

of software security practices and identify opportunities for refinement.

Participants The ten participants brought a wide range of professional experience to the interviews, offering rich insights into the integration of security practices within an Agile SDLC. As shown in Table 6.1, participants reported between 17 and 41 years of industry experience, with an average of 28.4 years. This breadth of experience highlights the depth and diversity of expertise informing the study's findings on adopting and operationalizing DAST in practice.

Data Analysis

Interview recordings were transcribed using the built-in transcription features of Microsoft Teams and Zoom to support systematic analysis. We applied an open coding approach, a common qualitative analysis technique [213], to examine the transcripts [214]. This process involved identifying and labeling concepts and grouping them into emergent themes without relying on predefined coding schemes. Two researchers independently coded the interview data and subsequently compared and reconciled their codes to arrive at a consolidated set of categories and interpretations.

An illustrative example of the data analysis process for RQ2 is shown in Table 6.3. In response to Q4 about challenges associated with integrating DAST into Agile workflows, participants highlighted multiple issues. For instance, participant T1 stated: *"One of the challenges was getting the DAST scans to work right. They are not automated in the deployment pipeline. JavaScript driver app payloads are garbage."* During coding, phrases such as "getting the DAST scans to work right" were categorized as *Setup Challenges*, while "not automated in the deployment pipeline" was coded as *Lack of Automation*. A similar process was applied across all interview responses, with statements often mapped to one or

Table 6.3: Open coding examples for "What are the general challenges you had while incorporating security practice into your Agile process?"

Participant	Response	Categories Identified
T1	"One of the challenges was getting the DAST scans to work right(Difficult in setup). They are not automated in the deployment pipeline. Javascript driver app payloads are garbage.DOM faced difficulty with ZAP and other initial tools. The biggest challenge going forward would be the disconnect between the modern web and Javascript. Artificial Intelligence (AI) will get into this too, there is future potential for an AI tool to parse web pages."	Setup Challenges, Tool Limitations, Lack of Automation, Disconnect with Modern Web (JavaScript), Future Solution: AI
T2	"Finding the right tool that we can work with. Our system had limitations. We could not create fake accounts for testing, and now it may be turned off."	Tool Compatibility, System Limitations (DUO 2FA), Testing Restrictions (No Fake Accounts)
T3	"Setting up a repeating card in an Agile board. Earlier upper management did not approve."	Cultural Challenges, Upper Management Approval
D1	"Finding the time bandwidth."	Time Constraints, Limited Bandwidth
D2	"I mean, there's this general challenge of parsing the report, but I don't think that's, you know, something you're not gonna be able to avoid."	DAST Report Parsing Challenges, Unavoidable Challenges
D3	"Since the assigned engineer did all of the work, and we did not have any vulnerabilities that needed to be fixed, there were no challenges for me. Once the assigned engineer is gone and if are to continue the DAST practice, then some of the challenges would be how to incorporate this into our pipeline, what is the right tooling, setting the balance between which levels of alert would need to be fixed and which to ignore."	No Challenges, Engineer Dependence, Future Challenges: Tool Selection, Pipeline Integration, Alert Prioritization
D4	"I did not have any challenges since we had you working on the setup and the configuration."	No Challenges, Engineer Dependence
I1	"I have had no general challenges while incorporating this security practice."	No Challenges
I2	"Did not encounter any challenges while incorporating this security practice into our SDLC."	No Challenges
I3	"I haven't had to do anything with myself, so it has not affected me except to check them out quarterly."	No Challenges, Minimal Involvement

more thematic categories. Due to IRB constraints, individual interview responses are not shared publicly.

6.3.5 Specifying Learning

Specifying Learning is the last stage that includes synthesizing broader lessons and insights derived from the evaluation phase [201]. Drawing on the findings from the interviews and analysis, Section 6.5 presents implications for improving DAST adoption and summarizes key lessons learned. These insights are intended to inform both practitioners and researchers who are considering or planning the integration of DAST and related security practices

within Agile development environments.

6.4 Results

This section reports the findings derived from the qualitative interviews.

6.4.1 RQ1: Willingness to Adopt and Continue DAST

As part of the interviews, participants were asked to reflect on their initial willingness to adopt DAST when it was first proposed, as well as their willingness to continue using it after it had been integrated into the team's workflow.

Initial Willingness

The interviews revealed a generally strong inclination among team members to adopt DAST as part of their Agile SDLC. Most participants expressed positive attitudes toward the introduction of DAST, viewing it as a meaningful addition to existing development practices. Among the ten interviewees, seven reported being **Very Willing** to adopt DAST, two indicated they were **Willing**, and one expressed an **Unwillingness** to do so. This unwillingness can be interpreted in the context of the team's existing high workload and the time pressures inherent in Agile development. Several participants emphasized the importance of security in modern software systems, particularly in contexts involving sensitive data. For example, I2 stated, *"Security is a critical aspect of software engineering, and incorporating DAST into our process seems like a natural step"*. Similarly, D4 highlighted the relevance of security when working with sensitive information, noting, *"Security is crucial, especially when dealing with large datasets that contain sensitive information"*.

Despite this overall enthusiasm, participants also identified a number of concerns that tempered their willingness to adopt DAST. Common issues included the anticipated complexity of setup, required resources, and organizational constraints. For instance, T1 described being **Willing** to adopt DAST but raised concerns related to **Political Factors**, specifically noting that **Obtaining Management Approval** could be challenging. Likewise, T2, while describing themselves as **Extremely Willing**, expressed uncertainty about the practical costs of adoption, stating concerns about *"how hard it is and what it will cost in money and man-hours"*. These perspectives illustrate how logistical and organizational considerations can moderate otherwise positive attitudes toward security tool adoption. Only one participant, D1, reported being **Unwilling** to adopt DAST, by explaining that they were *"concerned about return on investment and the development team's time and effort that would go into it"*. Such feedback suggests that while willingness to adopt DAST may be high overall, sustained adoption is likely to depend on clear value propositions and adequate organizational support.

Continued Willingness

Following the integration of DAST, all interviewed participants reported a strong willingness to continue using the tool. This included the developer D1 who had initially expressed unwillingness to adopt DAST. Their perspective shifted after observing a concrete security benefit during deployment: DAST identified a high-severity SQL Injection[9] vulnerability in one of the team's web services, which was subsequently prioritized and resolved quickly. This experience helped demonstrate the practical value of DAST and alleviated earlier concerns regarding return on investment and development effort. Of the ten interviewees, seven described themselves as **Very Willing** to continue, frequently citing DAST's ability to iden-

[9] SQL Injection

tify vulnerabilities early and its compatibility with existing Agile and CI/CD practices. For example, I2 remarked, *"I'm very willing to continue with DAST. It fits well with the DevOps philosophy of CI/CD"*. In a similar vein, I1 emphasized the practical security benefits, stating, *"I'm very willing to continue, especially since it has shown to be effective in identifying vulnerabilities that could compromise user identities"*. Observed technical benefits also influenced continued willingness, as illustrated by D4, who noted, *"I'm willing to continue with it, especially since we've seen benefits in identifying SQL injection vulnerabilities"*.

While no participant expressed an outright unwillingness to continue using DAST, several interviewees highlighted areas where improvements were needed. Four participants (T2, I2, D1, and D3) emphasized the importance of refining the integration to reduce developer effort and improve usability. In particular, D1 stressed the need for clearer and more actionable test outputs, suggesting that interpreting DAST results often requires specialized expertise: *"We need to review the set of tests that would generate alerts worth looking into. Need to be a security specialist to explain the reports"*.

Additional suggestions focused on increasing automation and deeper integration into CI/CD pipelines to minimize manual overhead. As D3 summarized, *"I am totally willing to continue, but the team has to decide. If the scans are integrated into CI/CD, it won't take much effort from the developers"*. Overall, these findings indicate a strong collective willingness to sustain DAST usage, coupled with a clear desire for enhancements that make the practice more efficient and less intrusive within the Agile development workflow.

6.4.2 RQ2: Perceived Impacts of DAST Integration

Impact on Day-To-Day Work

Participants were asked to describe how the introduction of DAST affected their daily work activities. All ten interviewees reported that DAST had either a **Low Impact** or **No Noticeable Impact** on their routine responsibilities, making this the most frequently observed response. This suggests that the integration of the security practice did not interfere with participants' core development or operational tasks. Most participants described their involvement with DAST as infrequent, typically limited to reviewing security reports on a quarterly basis, reflecting an overall pattern of **Occasional Involvement**. For example, D3 explained, *"It didn't really impact my day-to-day work at all, but again, that was because [assigned engineer] did all of the heavy lifting."* Similarly, D2 remarked, *"We only look at these things quarterly, almost none. 10–15 minutes every three months."*

These statements indicate that interaction with DAST was limited for most team members, either because few actionable vulnerabilities were identified or because security-related tasks were handled by designated individuals. Consequently, DAST required **Negligible Effort** from the majority of participants, with time investment largely restricted to brief reviews of scan results. As I1 described, *"I spend some time reviewing the scan results to ensure there are no identity-related vulnerabilities,"* highlighting that the effort primarily involved lightweight verification rather than sustained engagement.

Despite the minimal disruption, several participants reported a positive secondary effect in the form of **Increased Confidence in System Security**. This reassurance stemmed from knowing that no major vulnerabilities were being overlooked. As T2 noted, *"The positive impact is that we know there isn't any big vulnerability that needs to be taken care of, increasing our level of confidence in system security."* Taken together, these findings

suggest that DAST provided added security assurance while imposing minimal overhead on day-to-day development activities.

Challenges

Participants were also asked to reflect on any challenges they encountered while incorporating DAST into their workflow. Six of the ten interviewees indicated that they experienced **No Significant Challenges**, while the remaining participants reported only minor difficulties. This pattern suggests that, for many team members, the integration process was largely smooth, aided in part by the presence of a dedicated engineer responsible for setup and configuration. For example, D4 stated, *"I did not have any challenges since we had [assigned engineer] working on the setup and the configuration."* Similar views were expressed by I1, I2, and I3, who emphasized that task delegation reduced the burden of engaging directly with DAST.

This reliance on a small number of individuals reflects a broader pattern of **Engineer Dependence**, which helped mitigate short-term challenges but raises concerns about the long-term sustainability of the practice if key personnel become unavailable. Among participants who were more directly involved, **Time Management** and **Bandwidth Constraints** emerged as notable challenges. Both T3 and D1 reported difficulty balancing DAST-related activities with other development responsibilities, citing limited available time. In addition, D2 identified challenges related to **Report Interpretation**, explaining, *"There's this general challenge of sort of parsing the report, but I don't think that's something you're not going to be able to avoid."* Although technical challenges were less common than resource-related concerns, these responses highlight the need for improved automation and clearer reporting to further reduce friction for practitioners.

Impact on Team Velocity

To assess whether DAST affected development speed, participants were asked about its impact on team velocity. Eight of the ten interviewees reported that DAST had **No Impact** or only a **Minimal Impact** on overall velocity. The dominant perception was that, while DAST introduced some additional effort, it did not meaningfully slow down development. Several participants attributed this outcome to the fact that most DAST-related work was handled by the author/designated engineer. As D3 explained, *"There was no measurable impact of this practice on team velocity because 99.99% of the work was done by [assigned engineer]."* Others echoed this sentiment, noting that the allocation of dedicated resources helped shield the rest of the team from potential slowdowns. Additionally, I2 linked the minimal impact on velocity to the infrequent execution of scans, stating, *"It has a very minimal effect on the team velocity, as the scans themselves are conducted quarterly."*

While velocity remained largely unaffected, several participants expressed concern that security considerations were often secondary to task completion. This **Task-Oriented Mindset**, which prioritizes **Speed Over Security**, was described by I3, who noted that the team often aims to *"just sort of wanna get the job done and move the card from one column to the next."* Such comments point to a broader **Lack of Security Awareness**, where reliance on existing safeguards (e.g., firewalls) substitutes for a more comprehensive security approach. I3 further suggested that a cultural shift is needed to embed security more deeply into everyday practices, stating, *"If people got used to it and did it, I think it would just be part and parcel."* However, participants also acknowledged the difficulty of achieving such change within large organizations, as reflected in the observation that *"it's a big ship, and it's gonna turn slowly."* These remarks underscore the challenge of fostering a security-first culture amid organizational inertia, even when technical integrations have limited impact on productivity. Overall, the inclusion of DAST had little effect on team velocity, largely due

to the structure of the Agile process and the assignment of dedicated resources to manage security-related tasks.

Perceived Impact on Security

Participants were also asked whether they believed the software was more secure after DAST was incorporated into the Agile workflow. The majority of responses indicated a positive shift in perceived security. Eight of the ten interviewees explicitly stated that the product was **More Secure** following the introduction of DAST. One participant (D3) expressed uncertainty, suggesting that the benefits were indirect, while another (T2) felt that security levels remained largely unchanged.

Several participants highlighted specific benefits associated with DAST adoption, including **Increased Confidence in the Security Level** (T3, D3), **Increased Security Assurance** I2 and **Improved Awareness of the Current State of Security** (D2, T3). Many participants noted that DAST added an **Extra Layer of Protection** D4, especially for low-risk vulnerabilities, which were sometimes overlooked in their previous processes I1. As I2 explained, *"The DAST scans have identified vulnerabilities we might not have caught otherwise. These are critical components, and securing them has significantly reduced our risk profile and provided assurance in the security of our software systems."*

At the same time, some participants offered a more nuanced perspective, acknowledging that DAST alone does not provide comprehensive coverage. For example, D1 observed that *"the security scans are missing some class of errors,"* and suggested supplementing DAST with additional techniques such as AI- or ML-based analysis and fuzzing. Despite these limitations, most participants agreed that DAST had a beneficial impact on security. As D3 summarized, *"indirectly there is a security improvement anytime you introduce some*

software scanning process into your SDLC." Collectively, these findings indicate that DAST contributed to a stronger perceived security posture, while also highlighting the need for complementary tools and ongoing refinement.

6.4.3 RQ3: Improvements to DAST Integration

Participants were asked to suggest ways in which the integration of DAST and other security practices could be improved within Agile development processes. Several recurring themes emerged from the interview data, most notably the need for **More Frequent Testing**, increased **Automation**, and enhanced **Security Training**. Many participants emphasized that running security tests more regularly—and incorporating different types of tests, such as smoke and regression tests—would help identify vulnerabilities earlier in the software development lifecycle. As T1 observed, *"We need to move up the testing interval, i.e., make them more frequent,"* underscoring the perceived importance of consistent and timely security testing.

The need for greater **Automation** was highlighted by multiple participants, including T1, D4, and I3. Participants noted that automating DAST execution would reduce manual effort and ensure more consistent coverage. For example, I3 stated, *"Automation to run the scans whenever new releases are made should be a part of the CI/CD pipeline."* In a similar vein, D3 suggested that tighter integration with CI/CD tooling could further strengthen security enforcement, explaining, *"That would be the ultimate goal, which is to fail the pipeline if a vulnerability is found."* These comments reflect a shared view that automation is central to embedding security into routine development workflows.

Participants also emphasized the importance of **Better Feedback Loops** to better communicate security findings across the team. More frequent and accessible reporting was seen as

essential to keeping security concerns visible throughout the development life-cycle. Several interviewees pointed to the value of **Security Training** in strengthening the team's overall understanding of secure development practices. As I2 noted, *"Conducting regular security training for the DevOps team would help everyone understand the importance of security in the context of IAM."* In addition, I1 highlighted the potential benefits of more closely integrating security testing with **Identity and Access Management (IAM) Checks** to achieve broader coverage.

Further suggestions focused on **Reporting Improvements**, particularly the consistency of DAST results across different system components. For instance, D2 expressed interest in extending reporting beyond web interfaces, stating, *"Personally, I would like to see the same sort of reporting done against our REST APIs."* **Accessibility and Usability Concerns** were also raised regarding how results are accessed and reviewed. As D2 explained, *"I have no idea how to look at the HTML files. I have to download it and open it, which feels like an unnecessary extra step,"* highlighting the need for more user-friendly reporting formats.

Finally, some participants advocated for more advanced monitoring and analysis techniques to complement DAST. I3 suggested incorporating **Anomaly Detection** and **Log Monitoring**, as well as leveraging AI to support deeper analysis, noting, *"We should do more sophisticated data analysis on our logs; we could use AI to process these logs."* Collectively, these recommendations point toward a more proactive and integrated approach to security—one that combines frequent testing, automation, security training, and enhanced tooling to improve the effectiveness of DAST within Agile development environments.

6.4.4 RQ4: Comparing DAST with Other Security Practices

Participants were asked to compare DAST with other security practices they were familiar with and to reflect on the relative strengths of each approach. A prominent theme in the responses was that DAST provides **Runtime Vulnerability Detection**, which several participants identified as a key differentiator. Four interviewees emphasized that, unlike **SAST**, which analyzes source code to identify potential flaws, DAST evaluates applications while they are executing. This capability enables DAST to uncover vulnerabilities that only manifest at runtime, offering insights into how an application behaves under realistic conditions. As D3 explained, *"DAST is more live, done at runtime, which is fantastic because the static analysis does not give us runtime vulnerabilities."* For the team, this distinction was particularly valuable, as it allowed DAST to complement existing SAST practices by addressing security gaps that static analysis alone cannot cover. Participants further noted that integrating DAST alongside other testing approaches supports a **Layered Security Strategy**. By combining static and dynamic analysis techniques, teams can achieve broader coverage across both code-level and runtime vulnerabilities, thereby strengthening the overall security posture of the application. This multi-layered approach was viewed as especially important for identifying issues that might otherwise remain undetected if only a single security practice were employed.

In addition to its technical capabilities, participants highlighted the **Proactive** and **Collaborative** aspects of DAST, which contributes to **Improving Team Communication** and the overall security workflow. Several interviewees described how DAST facilitates **Continuous Monitoring** and encourages more effective communication around security issues within the team. For example, I3 emphasized the importance of maintaining visibility into discovered vulnerabilities, stating, *"We should track these vulnerabilities and notify the team regularly, being proactive."* Such practices help teams remain aware of emerging risks and

prioritize remediation efforts more effectively. Finally, participants observed that DAST works well in conjunction with other security mechanisms and hence **Complements Other Security Practices**, including access control and multi-factor authentication. As I1 noted, *"Security scanning, like DAST, complements these practices by identifying vulnerabilities that could potentially be exploited to bypass IAM controls."* By identifying application-level weaknesses, DAST enhances the effectiveness of existing security measures and was viewed by participants as a valuable addition to the team's overall security toolkit.

6.5 Discussion

6.5.1 Importance of Automation and Tooling

A central insight from this study is the pivotal role that automation plays in the successful integration of security practices within Agile workflows. Participants' willingness to adopt and continue using DAST reflects a growing awareness that security must be treated as an integral component of modern software development rather than an afterthought. Many interviewees also reported increased confidence in the security of the system after DAST was introduced, reinforcing the perceived value of automated security testing.

Lesson Learned: Automate Security Scans.

Prior to this initiative, the testing team relied largely on manual and ad hoc techniques to assess the security of their applications. These approaches were inefficient and difficult to scale to various web applications managed by the team, motivating the shift toward automated testing via DAST. Existing research suggests that security testing can impose significant cognitive burden on developers [215]. In contrast, automated tools have been shown to re-

duce cognitive load [216] while enabling effective and efficient vulnerability detection [217]. Consistent with these findings, our results suggest that Agile teams should integrate DAST scans directly into CI/CD pipelines to enable continuous assessment of security risks without requiring substantial manual effort.

Lesson Learned: Assign a Dedicated Engineer.

Participants consistently reported that DAST integration had little impact on their daily work, a result largely attributed to the presence of a dedicated engineer responsible for implementing and maintaining the security testing infrastructure. By centralizing responsibility for tool exploration and integration, the team was able to minimize disruption to ongoing development activities. Prior work has shown that developers often deprioritize security concerns due to lack of awareness or competing demands [218]. At the same time, studies have indicated that involving security specialists can significantly strengthen security outcomes in Agile settings [219]. Our findings support this view, suggesting that teams planning to adopt automated security testing—such as DAST—should consider allocating dedicated personnel to oversee integration and operation efforts.

Tool Recommendation: Enhanced Automation

The reliance on a single engineer, combined with participants' calls for improved automation, points to opportunities for advancing DAST tooling. Prior research has suggested that security practices are more likely to be adopted and sustained when they are highly automated and seamlessly embedded into development workflows [129]. Participants in our study highlighted limitations in existing tools, particularly related to report interpretation and the effort required to extract actionable insights. These observations suggest a need for more

advanced capabilities, such as automated report analysis, improved feedback mechanisms, and tighter integration with CI/CD and project management systems. Such enhancements could reduce manual overhead, improve usability, and further lower barriers to adopting security testing in Agile environments.

6.5.2 Balancing Speed and Security in Agile

Another prominent theme that emerged from this study is the inherent tension between Agile development principles and the demands of security testing. Agile methodologies prioritize rapid delivery, short feedback cycles, and continuous iteration, whereas comprehensive security practices often introduce additional complexity and overhead that can slow development [7]. This trade-off was reflected in participants' feedback, where team members consistently acknowledged the value of incorporating DAST while also recognizing the practical challenges associated with managing and acting on security scan results.

Lesson Learned: Be Flexible.

To accommodate both security and development velocity, the team integrated DAST into the CI/CD pipeline in a way that minimized manual intervention. Nevertheless, limitations related to tool capabilities and the effort required to interpret scan outputs persisted. Although prior research advocates for running automated tests frequently to maintain software quality [196], the team opted to execute DAST scans on a quarterly basis and to prioritize remediation of high-severity findings. This approach aligns with previous work suggesting that vulnerability prioritization can improve the security of web applications [220], and that infrequent automated testing may uncover new defects while repeated execution of the same tests can create a misleading perception of quality [221]. By adopting a flexible testing

strategy, the team was able to balance the goals of Agile delivery with the need to address security risks. These findings suggest that Agile teams should adapt the frequency and scope of automated security testing to their specific context rather than adopting a rigid, one-size-fits-all approach.

Tool Recommendation: Enhance Tool Output.

Consistent with Chapter 1, this study revealed that developers faced persistent challenges in interpreting and effectively acting on the outputs produced by DAST tools. Prior research has also indicated that understanding security reports can be challenging for practitioners and may hinder effective remediation [28]. As we saw previously, Chapter 5 explored approaches to address this issue, by using LLMs to summarize and contextualize DAST reports in more accessible, human-readable formats. Building on these efforts, future research and tool development should further investigate techniques to simplify and contextualize automated security reports, thereby reducing cognitive burden and supporting more efficient decision-making in Agile development teams.

6.5.3 Cultural Shift Towards Security Awareness

The findings of this study point to the importance of fostering cultural change in how development teams perceive and prioritize security. Although the team explicitly scheduled security testing and DAST-related tasks on their Kanban board, many participants continued to view security as secondary to the primary objective of delivering features quickly. Additionally, organizational challenges—such as *Internal Politics* and the need to *Obtain Management Approval*—were cited as factors that reduced initial willingness to adopt DAST. These observations suggest that, despite formal process changes, underlying attitudes and incentives

can still limit the extent to which security is fully embedded into everyday development practices.

Lesson Learned: Foster a Security-First Culture

Prior research indicates that software engineers often assign lower priority to security concerns [218], and that tensions may exist between development teams and security specialists [222]. Other studies highlight the role of organizational culture in shaping the adoption and effective use of security tools [125]. Our findings reinforce these insights, suggesting that strengthening a security-oriented culture can meaningfully influence how security practices are adopted in Agile settings. Investing in tools that surface clear, actionable insights from security scans can help reinforce the importance of security in daily workflows. For example, participants observed that the commercial version of Burp Suite offered enhanced functionality and support compared to the open-source ZAP tool, which influenced perceptions of tool effectiveness.

Because Agile development emphasizes frequent feedback [223], regularly reviewing security findings and sharing concise, understandable feedback with all stakeholders can help ensure that security remains visible and prioritized [224]. Prior work also suggests that increased communication and collaboration—both with management and among developers—can improve security practices within development teams [125]. Together, these findings indicate that cultivating a security-first mindset requires not only technical solutions but also sustained organizational commitment and communication.

Tool Recommendation: Support Security Culture

Participants offered several suggestions for tooling enhancements that could reinforce a security-oriented culture. These included clearer, more understandable, and more accessible reporting, improved feedback loops to facilitate discussion of security issues, and greater support for automation and training related to DAST integration. Participants also noted that identifying appropriate DAST tools was sometimes unintuitive; in one instance, the team had to restart the selection process after discovering that a chosen tool did not meet requirements due to outdated policy support. As security practices become increasingly automated, tools can incorporate features specifically designed to promote security awareness within development teams. Examples include regular security workshops [225, 226], gamified security challenges [227], using LLMs to summarize issues and provide more actionable security reports (Chapter 5) and the integration of lightweight security checklists into development workflows [228]. Collectively, such approaches can help normalize security considerations and embed them more deeply into Agile development culture.

6.5.4 Comparative Strengths and Gaps of DAST

Our findings indicate that DAST provides distinct advantages when compared to other security practices, most notably its ability to uncover vulnerabilities that arise during application execution and may not be detected through static analysis alone. By operating at runtime, DAST offers an additional layer of protection that complements code-focused techniques. At the same time, participants also identified important limitations of DAST. In particular, they noted that certain classes of vulnerabilities—such as logical flaws or business logic errors—are difficult for DAST tools to detect, as these issues often require deeper contextual understanding of application behavior and requirements.

Lesson Learned: Adopt a Layered Security Approach

To address the limitations of individual security techniques, teams can benefit from combining multiple security practices. For example, DAST can be used alongside other approaches [229], such as SAST [103, 230], to achieve broader and more comprehensive vulnerability coverage. A layered strategy that integrates several tools and methods enables teams to address both pre-deployment and post-deployment security concerns. Prior research similarly advocates for holistic security strategies that incorporate static and dynamic analysis techniques [231], as well as continuous monitoring and ongoing security training, to strengthen overall system resilience.

Tool Recommendation: Complement Other Security Practices

Interview participants also highlighted the value of DAST tools that integrate smoothly with other security mechanisms, including IAM-related analyses, multi-factor authentication, and SAST solutions. This interoperability was viewed as a key strength, as it allows DAST to enhance, rather than duplicate, existing security efforts. Previous studies have shown that poor compatibility between security tools and established development workflows is a significant barrier to adoption [232]. Consequently, designers of automated security tools should prioritize compatibility and integration with a broad range of security and development practices to facilitate adoption and sustained use in Agile environments.

6.6 Limitations

This study on integrating DAST in Agile was subjected to several threats to validity that should be considered when interpreting the results.

Internal validity Internal validity concerns stemmed primarily from the reliance on qualitative data gathered through interviews and observational methods. Such data may be influenced by participant bias, social desirability effects, or inaccuracies in recall. In addition, the researcher's embedded role within the team may have introduced observer bias, whereby participants altered their behavior due to the presence of the researcher. To mitigate these risks, interviews were assured anonymity, and multiple researchers—including one external to the team—were involved in coding and analyzing the interview transcripts to reduce individual bias.

External validity External validity is constrained by the limited scope of the study, which focused on a single case study team integrating ZAP and Burp Suite within a Kanban-based Agile process and using GitLab for CI/CD automation. As a result, the findings may not readily generalize to other DAST tools (e.g., Veracode[10]), alternative development methodologies (e.g., Scrum), different CI/CD infrastructures (e.g., GitHub Actions[11], Travis CI[12]), or organizations operating in different domains with varying team structures, cultures, or constraints. Although the case study offers detailed insights into DAST adoption in a specific context, broader generalization would require replication across multiple teams and diverse environments.

Construct & Conclusion validity The interpretation of security integration outcomes is inherently subjective, and the relatively small number of interview participants may limit the robustness of the conclusions. While open coding was employed to systematically analyze qualitative data, variations in participants' perceptions of what constitutes successful security integration and potential selection bias may influence the results. Moreover, this study

[10] https://www.veracode.com/products/dynamic-analysis-dast/
[11] https://github.com/features/actions
[12] https://www.travis-ci.com/

primarily examines practitioners' perceived impacts of DAST integration rather than directly measuring changes in software security outcomes. Future work should therefore complement qualitative insights with quantitative analyses to evaluate the concrete effects of DAST on the security of software products.

6.7 Future Work

There are several promising directions for extending this work. Future research can broaden the scope by examining developer perceptions and experiences with security practices beyond DAST and across development methodologies other than Kanban. For example, studying how security activities such as SAST, threat modeling, dependency vulnerability scanning, log-based fraud detection, etc., are integrated into development frameworks like Extreme Programming (XP), Scrum, or other hybrid Agile approaches could reveal effective integration strategies. Such studies would provide a more comprehensive understanding of how different security tools and practices can be adapted to diverse team structures and development workflows.

In addition to methodological diversity, future work could incorporate quantitative measures to complement the qualitative insights presented in this study. To evaluate the effect of DAST integration in Agile workflows objectively, organizations can track key performance indicators like vulnerability count and severity levels, remediation timelines, and false-positive frequency, alongside shifts in team velocity and overall productivity. Combining qualitative and quantitative data would enable a richer evaluation of both perceived and measurable impacts of security tooling. Long-term sustainability is another important area for future investigation. Longitudinal studies that track software teams over extended periods could shed light on how security practices like DAST evolve after initial adoption,

whether they remain embedded in daily workflows, and how the teams adapt their use of these tools as systems and organizational contexts change. Expanding research to include multiple teams across different organizations and industries would further support generalization of findings and help identify best practices tailored to varying security requirements and constraints. Finally, future work can explore advanced techniques to enhance the integration and usability of DAST within development pipelines. In particular, leveraging AI and LLMs offers opportunities to improve vulnerability detection [233] and to generate concise, human-centric summaries of security findings that are easier for developers to interpret and act upon (Chapter 5). Such approaches have the potential to reduce cognitive overhead, improve decision-making, and further align security practices with Agile development principles.

6.8 Conclusion

This chapter examined the adoption of DAST in Agile-Kanban workflows via an in-depth case study conducted within a real-world software team. The findings indicate that integrating DAST into CI/CD and Kanban processes can be achieved with minimal disruption to team velocity, particularly when supported by automation and the involvement of dedicated personnel. Most participants expressed both an initial willingness to adopt DAST and a strong inclination to continue using it, largely due to its ability to surface security vulnerabilities without imposing substantial overhead on their daily development activities. The study further shows that DAST integration aligns well with iterative development practices commonly found in CI/CD pipelines. Participants reported reductions in manual security effort and an overall increase in confidence in the security posture of the system following adoption. At the same time, the findings highlight several challenges, including difficulties in

interpreting security reports, prioritizing identified vulnerabilities, and the need for deeper automation to further streamline integration. Taken together, these results provide actionable insights for both practitioners and tool developers. They underscore the importance of carefully balancing rapid software delivery with the imperative of building systems that are both secure and robust. In addition, the results point to opportunities for improving automated security tooling to better support Agile teams in integrating security practices into their everyday workflows.

Chapter 7

SafeAIMerge: A Security Tool for Integrating DAST and LLM Feedback into GitHub Workflows

7.1 Motivation

Modern software development increasingly relies on Agile methodologies, which emphasize rapid, incremental delivery, and on CI/CD practices that operationalize these principles through frequent updates, automated testing, and continuous deployment [2, 33, 34]. While these approaches enable speed and adaptability, integrating effective security practices into such lightweight workflows remains a persistent challenge [39, 150]. In particular, security activities that introduce additional complexity, delay feedback, or require specialized expertise are often deprioritized in favor of rapid feature delivery [234, 235]. For instance, DAST tools, such as ZAP[1] and Burp Suite[2], are used to simulate attacker behavior and detect runtime vulnerabilities in web applications [18, 172]. However, their adoption in Agile teams is often hindered by multiple factors. First, the reports they generate are lengthy, technical, and difficult for non-specialists to interpret [20, 236]. Second, their integration into developer-

[1] https://www.zaproxy.org/
[2] https://portswigger.net/burp/pro

centric workflows, like GitHub PRs, is limited [129]. Moreover, because DAST tools operate dynamically and externally, they typically lack direct awareness of the static code changes that introduced a vulnerability, making it harder for developers to relate reported issues back to specific modifications in a PR.

Chapter 5 showed developers struggle with cumbersome and complex terminology of DAST security reports. These barriers are exacerbated in Agile contexts, where minimizing documentation and supporting rapid feedback cycles are core principles [148, 149]. LLMs offer promising capabilities for making technical reports more accessible and usable. Accessibility is an important consideration given that complex and verbose security outputs often hinder developers' ability to engage, understand, prioritize, and act on identified issues [237]. Moreover, the psychological aspects of usability in security systems can significantly influence developers' engagement with security warnings. Lennartsson et al. [238] assert that user reluctance to utilize security solutions is a common hindrance. Therefore, improvements in usability can lead to better compliance with security protocols and ultimately enhance protection against security threats. This suggests that effective communication of security reports must address not only the content but also the format in which this information is presented. LLMs have demonstrated strong performance in summarization and generating domain-specific natural language guidance [108, 115, 177]. Applied to security, they can simplify vulnerability descriptions, reduce jargon, and provide security summaries tailored for developers (Chapter 5). Embedding such summaries directly into GitHub workflows has the potential to lower the barrier for adopting automated security testing without requiring specialized expertise.

To better understand how recent advances in LLMs have been applied within cybersecurity workflows, we conducted a literature survey, the results of which are recorded in Chapter 2 Section 2.4. This revealed that while LLMs have been explored extensively for vulnerabil-

ity detection, static analysis augmentation, penetration testing assistance, and secure code generation, most existing approaches either operate outside of developers' primary workflows or target automated detection and repair rather than developer-facing communication. To the best of our knowledge, no prior work integrates DAST scanning with LLM-based summarization and remediation guidance in a lightweight, CI/CD-compatible manner that embeds security feedback directly into GitHub PR workflows. This gap motivates our focus on improving the usability and actionability of security reports at the point where developers already collaborate and make code changes.

As a solution, we present *SafeAIMerge*[3], a lightweight CI/CD-integrated security tool that combines DAST scanning with LLM-based summarization and remediation guidance to provide developer-centric security feedback within GitHub PR workflows. *SafeAIMerge* integrates the ZAP[4] tool into GitHub Actions[5] pipelines and is triggered automatically when a PR is created or updated. As a secondary benefit, by providing the LLM with both, the DAST alerts and the code changes introduced in the PR, the LLM produces summaries and remediation steps that are directly tied to the code changes being reviewed. Hence, helping offset DAST's limited view of the source code.

When a PR is created or some change is pushed, *SafeAIMerge* automatically:

1. Executes DAST scans on both the `main` branch and the PR branch versions of the application.
2. Compares the resulting security reports to identify new, resolved, and existing alerts.
3. Routes prioritized alerts along with PR code changes to an LLM to generate per-alert summaries and actionable remediation guidance.

[3] https://github.com/arpitthool/SafeAIMerge
[4] https://www.zaproxy.org/
[5] https://github.com/features/actions

4. Compiles a consolidated security report, including an overall synopsis and severity-based breakdowns.

5. Delivers the results through an enhanced PR comment with structured dropdowns and an HTML report artifact link for detailed review.

SafeAIMerge workflow aligns closely with the core Agile principles that emphasize rapid feedback, continuous integration, and close collaboration within lightweight development processes. Agile methodologies prioritize frequent integration of changes and early identification of defects to maintain development momentum and deliver functional software in short cycles [34, 239]. By automatically executing DAST scans whenever a PR is created or updated, *SafeAIMerge* embeds security checks directly into CI/CD pipelines, enabling developers to detect and address vulnerabilities incrementally rather than deferring security activities to later stages [19]. This approach reinforces Agile's emphasis on timely feedback and supports secure, customer-centric delivery without introducing additional process overhead. Furthermore, *SafeAIMerge* operationalizes Agile values of collaboration and responsiveness by presenting contextual security summaries and remediation guidance as structured PR comments. These promote shared understanding, support severity-based prioritization, and enables rapid discussion within existing collaboration spaces [240, 241]. By minimizing context switching and seamless GitHub integration, *SafeAIMerge* helps in shortening the security feedback loops in development workflows, a key requirement for Agile [242, 243].

To systematically evaluate the effectiveness of *SafeAIMerge* as a developer-facing, LLM-assisted security tool, we formulated the below research questions (RQs). These RQs were selected to reflect the core challenges identified in prior chapters and to focus on outcomes that are critical for the practical adoption of security tools in Agile and CI/CD environments, including developer workload [244], vulnerability remediation effectiveness [245], and

workflow efficiency [246].

RQ1 How does *SafeAIMerge's* security feedback affect developer workload during vulnerability remediation compared to a baseline DAST workflow? *This question investigated whether summarizing and contextualizing DAST findings within PR can reduce the cognitive and emotional burden associated with interpreting and acting on security reports.*

RQ2 Does *SafeAIMerge's* security report improve developers' ability to understand, remediate, and efficiently resolve vulnerabilities compared to traditional DAST reports? *This question examined whether developer-oriented summaries and remediation guidance enable developers to more effectively translate security findings into concrete fixes.*

RQ3 How does the *SafeAIMerge* workflow compare to a traditional DAST workflow in terms of developer workflow preference when addressing security vulnerabilities in CI/CD pipelines? *This research question focuses on developers' comparative workflow preferences after experiencing both approaches, capturing perceived usability, clarity, and overall workflow fit.*

To evaluate *SafeAIMerge*, we conducted a two-phase empirical evaluation. In the first phase, we carried out an online survey with 46 industry practitioners to capture initial perceptions of usability, clarity, and adoption potential. The survey results indicate that developers prefer LLM-generated summaries over raw DAST reports and value the integration of LLM-generated security feedback directly into GitHub PRs. This phase also served as a formative study informing us about iterative refinements to the tool's reporting and presentation prior to conducting a controlled evaluation. The second phase, which was explicitly designed to answer the above research questions, we conducted a controlled, task-based user study with

12 practitioners, comparing the *SafeAIMerge* workflow with the baseline ZAP workflow. Using NASA Task Load Index (NASA-TLX) workload measures, task completion outcomes, and comparative preference data, this study provides a direct empirical insight into how *SafeAIMerge* reduces developer workload, increases the effectiveness to resolve security issues, and improve the overall experience when addressing security vulnerabilities in CI/CD pipelines.

In the remainder of this chapter, we describe the design, implementation, and evaluation of *SafeAIMerge*. We first present the design rationale and core principles that guided the development of the tool (Section 7.2), followed by a detailed description of its system architecture and implementation (Section 7.3). We then report the results of our two-phase empirical evaluation (Section 7.4). Finally, we discuss the limitations of our evaluation (Section 7.5), outline directions for future work (Section 7.6), and conclude the chapter by summarizing the key findings and contributions of *SafeAIMerge* (Section 7.7).

7.2 Design

The design of *SafeAIMerge* was directly informed by the empirical insights and design implications derived from previous chapters. Chapter 4 established that Agile practitioners are generally willing to adopt security practices when they are automated, iterative, and integrated into existing CI/CD workflows, and that such practices do not substantially harm perceived productivity when friction and overhead are minimized. These findings motivated *SafeAIMerge's* emphasis on fully automated DAST execution within PR workflows, aligning security checks with natural developer activity rather than introducing parallel processes. Insights from Chapter 5 informed *SafeAIMerge's* focus on improving the accessibility and actionability of security feedback through LLM-generated summaries. Chapter 6 further

reinforced the importance of automation and tooling in sustaining security practices in Agile settings, highlighting that minimizing manual effort, improving the actionability of tool outputs, and reducing disruption to day-to-day development are critical for continued adoption. This directly guided *SafeAIMerge's* use of LLM-based summarization and remediation guidance to transform verbose scan outputs into concise, developer-oriented feedback. Based on these insights, we derived four core design principles for *SafeAIMerge*:

1. **Automation**, to ensure security checks occur consistently and early.
2. **Developer-centric integration**, to embed security feedback within existing workflows.
3. **Developer-centric reports**, to reduce cognitive burden and improve comprehension.
4. **Lightweight adoption**, to minimize configuration and maintenance overhead.

These principles informed the key features described below, grounding the tool in both empirical evidence and practical usability considerations.

1. **Automated Security Scanning in PRs:** To ensure that security checks occur without requiring manual intervention, *SafeAIMerge* automatically triggers a DAST scan using ZAP whenever a PR is created or code is pushed to an existing one. Past research [64, 134] highlights the value of iterative and incremental vulnerability scanning in Agile workflows, where automation reduces friction and increases the likelihood of adoption. Agile methodologies emphasize rapid feedback and iterative development, which are operationalized through CI/CD pipelines that automate integration, testing, and deployment [2, 33, 34]. DevSecOps builds on this Agile–CI/CD foundation by embedding security controls directly into automated pipelines, enabling continuous and early security feedback rather than late-stage validation [247, 248].

2. **LLM-Assisted and Context-Aware Summarization of Security Alerts:** Chapter 5 demonstrated that practitioners find traditional DAST reports verbose and difficult to comprehend, and strongly prefer concise summaries generated by LLMs. To reduce the cognitive load on developers, *SafeAIMerge* routes prioritized DAST alerts through an LLM that produces short descriptions of each issue, recommended remediation steps, and high-level summaries. Additionaly, the LLM prompts for newly introduced alerts—i.e. alerts that are introduced by the addition of the PR code—are also enriched with the PR code-diff information.

   During early iterations of *SafeAIMerge*, the LLM was provided only with raw DAST alerts. While this often produced generally reasonable remediation advice, the suggestions were frequently abstract and not clear enough to operationalize because the model had no visibility into the code under review. To assess whether PR code context improves actionability, we performed a controlled comparison in which we generated two LLM-based security reports for a sample PR in a GitHub repository[6]. The PR contained eight new security alerts instances—6 medium-level and 2 low-level. In the "without code-diff" condition, the LLM prompt contained only the DAST alert along with prompt text; in the "with code-diff" condition, we provided the same alert along with the PR's code-diff along with the same prompt text.

   We evaluated each condition manually by applying the corresponding LLM-generated remediation steps exactly as described onto the PR branch, without adding additional interpretation or external guidance. After applying the suggested fixes, we re-ran *SafeAIMerge* on the updated branch using the same scan workflow. We considered an alert successfully resolved if the corresponding alert instance was no longer reported in the follow-up *SafeAIMerge* scan. This was repeated three times for both conditions.

[6] https://github.com/arpitthool/pac-man

Across all three runs, for the eight new alert instances, the "with code-diff" reports consistently produced more concrete and in-context guidance: they identified which file to modify and proposed explicit code-level changes (add/remove/modify) rather than only describing a general mitigation strategy. Consistently across the three runs, implementing these "with code-diff" recommendations resolved 7/8 alert instances; the remaining one alert persisted despite implementing the suggested change. In contrast, across all three runs, the "without code-diff" reports produced high-level best-practice recommendations and supplemented them with external reference links (e.g., web documentation such as MDN/OWASP/CWE), which were not visited or consulted during remediation. These alerts did not explicitly indicate where in the codebase to apply the fix. None of them had a direct, file-specific solution described in the report. For example, as shown in Figure 7.1 & 7.2, for the *"Server Leaks Version Information via Server HTTP Response Header Field"*[7] alert, the "with code-diff" report pointed to the relevant file in the PR and recommended adding specific code/configuration, whereas the "without code-diff" report only described the generic mitigation and provided web documentation links.

We limited this analysis to three runs because the evaluation required manually reviewing each alert and applying the remediation steps exactly as described, which does not scale well to a larger alert set. These preliminary findings suggest that providing PR code-diff context enables the LLM to generate more actionable remediation guidance for new alerts. This approach operationalizes calls for more developer-centric and context-aware security reporting [28, 236], resulting in summaries that are easier for non-specialists to understand and more readily actionable within developer workflows.

3. **Seamless GitHub Integration:** Developers frequently resist adopting tools that

[7] Server Leaks Version Information via Server HTTP Response Header Field

Alert: Server Leaks Version Information via "Server" HTTP Response Header Field **(Low Risk)**

Summary
Our web application server is leaking version information through the "Server" HTTP response header. This could potentially allow attackers to identify other vulnerabilities in our server. The issue was identified at the following node: http://my-web-app-pr:3000/feedback.html. The confidence level of this alert is high, but the risk is low.

Actionable Steps

(a) Review the server configurations to ensure that the "Server" header is either suppressed or provides only generic details. This can be done in the server configuration file (like `nginx.conf` for an Nginx server).

(b) Considering the recent PR changes, specifically in the `feedback.html` and `nginx.conf` files, ensure that any new configurations or changes do not expose version or other sensitive information.

(c) After making the necessary changes, re-run the security scan to confirm that the issue is resolved.

(d) Keep monitoring the server logs and security alerts for any potential threats.

Suggested Code Changes
In the `nginx.conf` file, add or modify the following line to prevent the server from revealing its version:

```
server_tokens off;
```

In the `feedback.html` file, ensure that no sensitive information is being exposed through the HTTP headers or elsewhere. Remember to test these changes thoroughly to ensure they don't affect the functionality of the application.

Figure 7.1: Example of LLM-generated remediation guidance *with PR code-diff* context. The output explicitly references modified files and proposes concrete configuration-level changes.

require switching contexts or operating outside their existing environments [129]. To address this, *SafeAIMerge* integrates directly with GitHub by delivering security feedback within the PR interface. This ensures that security insights are delivered in the same interface where code reviews and discussions already take place. The tool generates PR comments that organize findings into structured, interactive sections. Specifically, alerts are grouped into separate dropdowns for newly introduced, resolved, and existing vulnerabilities, with each group further organized by severity level and summarized individually. In addition to inline PR feedback, *SafeAIMerge* produces an HTML security report that serves as a downloadable artifact. This report provides alert-based summaries and fix-recommendations, interactive elements, and access to full vulnerability information for deeper inspection. Together, these mechanisms balance developers' need for lightweight, in-context feedback with organizational needs

Alert: Server Leaks Version Information via "Server" HTTP Response Header Field **(Low Risk)**

Summary
The web/application server at http://my-web-app-pr:3000/feedback.html is leaking version information via the "Server" HTTP response header (nginx/1.29.4). This security misconfiguration poses a low-risk threat but has a high confidence level.

Actionable Steps

(a) Reconfigure your web server, application server, or load balancer to suppress the "Server" header or provide generic details only.

(b) Review the related OWASP guidelines and the provided references for further understanding and prevention methods:

- OWASP Top 10 A05: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/
- OWASP A6 2017: https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration.html
- Web Security Testing Guide: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/01-Information_Gathering/02-Fingerprint_Web_Server
- CWE-497: https://cwe.mitre.org/data/definitions/497.html

(c) Refer to the following guides for specific server configurations:

- Apache: https://httpd.apache.org/docs/current/mod/core.html#servertokens
- Microsoft: https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff648552(v=pandp.10)
- Guide on response headers: https://www.troyhunt.com/shhh-dont-let-your-response-headers/

Please ensure these steps are taken promptly to prevent potential attackers from identifying other vulnerabilities.

Figure 7.2: Example of LLM-generated remediation guidance *without PR code-diff* context. The output provides general best-practice mitigation advice and external reference links.

for comprehensive and auditable security documentation.

4. **Lightweight Adoption and Workflow Fit:** In Chapter 4 the survey of Agile practitioners found that teams are most willing to adopt security tools when they fit seamlessly into CI/CD pipelines and do not demand significant workflow changes, which was also resonated in Chapter 5. Accordingly, *SafeAIMerge* was designed as a plug-and-play tool that requires minimal configuration. The tool integrates natively with GitHub Actions[8] and is packaged as a self-contained `.security` directory that can be copied into an existing repository. Adoption typically involves adding this directory and making small adjustments to a `YAML` configuration file to specify scan policies and

[8] https://github.com/features/actions

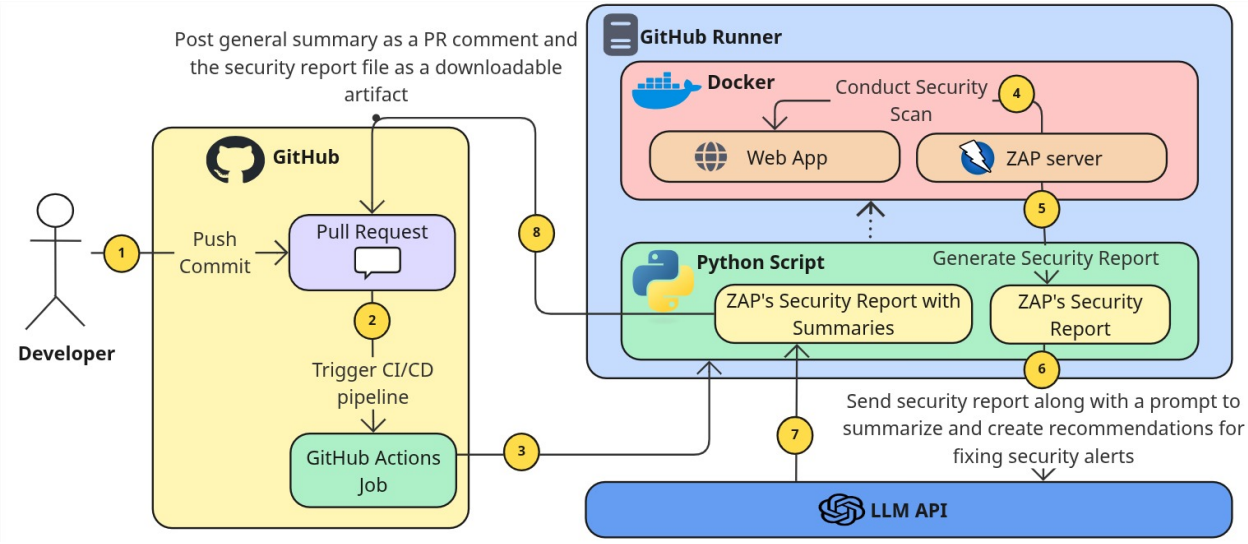


Figure 7.3: System architecture and dataflow for *SafeAIMerge*. The numbered arrows (1–8) indicate execution order on PR updates.

thresholds. In addition, the tool uses Docker[9], which further streamlines deployment across heterogeneous environments by encapsulating dependencies and configurations. This guarantees consistency of execution across machines, reduces environment-specific errors, and simplifies scaling and maintenance. By leveraging containerization alongside native CI/CD integration, *SafeAIMerge* reduces setup and maintenance overhead, thereby lowering the barrier to adoption in fast-paced Agile teams.

7.3 Implementation

The implementation of *SafeAIMerge* is organized as a modular CI/CD pipeline integrated with GitHub PR workflows. The system is designed to be adaptable across diverse projects while maintaining a lightweight footprint suitable for Agile teams. Figure 7.3 illustrates the overall system architecture and execution flow, which we describe in detail below.

[9] https://www.docker.com/

1. **CI/CD Integration with GitHub-Actions:** At the core of the system is a GitHub Actions workflow file (`zap.yaml`)[10] that orchestrates the execution of DAST scans and subsequent analysis steps in response to developer activity. The workflow is automatically triggered under three conditions.

   (a) When a new PR is created or updated against the `main` branch.

   (b) During scheduled execution of cron-job[11] to support periodic security assessments.

   (c) Manual invocation through the GitHub Actions interface for on-demand scans.

   Upon being triggered, the workflow starts on an available GitHub-runner, launches the target web application or API within a Docker container, and starts a ZAP server container configured with the required API access keys. This containerized setup ensures reproducibility, isolation, and consistency across executions. Once the environment is initialized, the workflow installs required dependencies, executes the *SafeAIMerge* scanning and analysis scripts, and publishes the generated security artifacts to GitHub.

2. **Dual-Branch Scanning and Comparative Analysis:** *SafeAIMerge* performs two coordinated scans per PR. One scan targets the application corresponding to the `main` branch, while the second scan targets the application version built from the PR branch. This dual-scan approach enables *SafeAIMerge* to reason about security changes introduced by the PR rather than presenting vulnerabilities in isolation. After both scans complete, the resulting alert sets are compared to classify vulnerabilities into three categories: (i) newly introduced alerts by the PR, (ii) resolved alerts by the PR, and (iii) existing alerts that remain unchanged. This comparison allows *SafeAIMerge* to explicitly surface the security impact of code changes, supporting developer decision-making

[10] zap.yaml
[11] https://en.wikipedia.org/wiki/Cron

during code review and aligning security feedback with the semantics of PR-based development.

3. **Configurable Scan Policies:** To support varying security requirements and risk tolerances across projects as observed in Chapter 6, *SafeAIMerge* relies on a YAML-based configuration file (`config.yaml`)[12] that decouples scan policies from workflow logic. This configuration allows developers to specify:

   - Which ZAP scans to enable (e.g., Spider,[13] AJAX Spider,[14] Passive scan,[15] Active scan[16]).
   - Risk levels to include in summaries (e.g., `High`, `Medium`, etc.).
   - Risk levels to ignore (e.g., `Low`, `Informational`, etc.).
   - Severity thresholds that fail the CI pipeline (e.g., blocking merges on `High`-severity alerts).
   - The maximum number of alerts included in a single report.

   By externalizing these options, *SafeAIMerge* remains lightweight to adopt and easily adaptable to different team policies without requiring changes to the core implementation.

4. **Automated Scanning and Alert Collection & Comparison:** The primary scanning logic is implemented in `scan.py`,[17] which communicates with ZAP to manage scan execution and data collection. Specifically, the script:

   (a) Launches the configured ZAP scans for `main` and PR branch instances.

[12] config.yaml
[13] ZAP Spider scan
[14] ZAP AJAX spider scan
[15] ZAP Passive scan
[16] ZAP Active Scan
[17] SafeAIMerge Scan Python Script

(b) Monitors scan progress until completion.

(c) Retrieves the complete set of detected alerts from ZAP for each scan.

All communication occurs over the private Docker network shared by the target application containers and the ZAP container, ensuring that the scanner can fully explore the running application in a realistic deployment context. Once alerts are collected from both scans, *SafeAIMerge* sorts them by severity level. Then it matches alerts across scans to determine whether vulnerabilities are newly introduced, resolved, or unchanged. This ensures that subsequent reporting focuses developer attention on the most relevant security changes associated with the PR.

5. **LLM-Based and Code-Aware Summarization:** Detected and prioritized alerts are processed by `alert_processor.py`[18], which implements a configurable filtering and summarization pipeline. Alerts that do not meet configured severity thresholds are discarded, while pipeline-blocking alerts are flagged explicitly. For the remaining alerts, structured JSON representations are passed to an LLM (OpenAI[19] in our prototype) using tailored system prompts. In case of new alerts, *SafeAIMerge* enriches LLM prompts with contextual information from the PR, including code-diffs and alert comparison results. This additional context enables the model to generate more targeted summaries, actionable remediation guidance, and explanations that directly relate vulnerabilities to the code changes under review. The output includes per-alert summaries, recommended fixes, and a high-level synopsis of the security impact of the PR.

6. **Report Generation and Visualization:** *SafeAIMerge* generates a consolidated HTML report, providing severity-based summaries and interactive elements such as

[18] SafeAiMerge Alert Processor Python Script
[19] `https://openai.com/index/openai-api/`

expandable sections for individual alerts. This report enables deeper inspection of findings without overwhelming developers during routine reviews. This report is archived as a downloadable artifact for traceability and offline viewing. We also archive the raw file which contains the alerts in JSON format.

7. **Feedback in GitHub PRs:** To deliver security feedback directly within the developer workflow, *SafeAIMerge* uses the GitHub API[20] to post a structured comment on the PR along with the link to download the full security report. These comment include interactive dropdown sections that separately summarize newly introduced alerts, resolved alerts, and existing alerts, each organized by severity levels. By embedding this feedback into the PR discussion, *SafeAIMerge* minimizes context switching and ensures that security considerations are addressed alongside functional code review.

8. **Extensibility and Pipeline Control:** To align with DevSecOps adoption practices, *SafeAIMerge* was implemented with extensibility in mind. Prompts for the LLM are externalized as text files, allowing teams to customize the language, level of detail, or remediation style without altering the core code. Additionally, pipeline gating rules ensure that critical vulnerabilities can block merges, supporting teams with stricter compliance requirements.

7.4 Evaluation

To evaluate *SafeAIMerge*, we conducted a two-phase evaluation. First, we carried out an online survey with industry practitioners to capture broad perceptions of usability, clarity, and workflow fit, and to gather early feedback that informed subsequent design refinements. Second, we conducted a controlled, task-based user study to empirically assess how

[20] GitHub Rest API

SafeAIMerge affects developer workload, task performance, and vulnerability remediation effectiveness when compared to a baseline DAST workflow. Together, these complementary evaluations provide both perception-level and behavior-level evidence of the tool's impact in CI/CD settings. Both the survey and the user study were approved by our Institutional Review Board (IRB).

7.4.1 Phase I: Online Practitioner Survey

Survey Design

The preliminary survey captured both quantitative and qualitative perceptions of *SafeAIMerge*, providing participants with an overview, a video demonstration of its workflow and configuration options, and a GitHub repository link[21] for further exploration. It consisted of twelve questions: five Likert-scale items, two binary questions, three open-ended prompts, and two categorical items. The Likert-scale questions measured overall user-friendliness, ease of integration into CI/CD pipelines, usefulness of GitHub PR comment integration, likelihood of adoption, and overall satisfaction. Binary and categorical questions asked whether LLM-generated summaries improved understanding, whether participants would recommend the tool, and how easily it could be integrated into existing workflows. Open-ended questions solicited feedback on confusing aspects of the UI, obstacles to adoption, compelling features, desired improvements, and any additional comments; these questions were optional. The survey questions are provided in Appendix C.1. This mixed-method design enabled a holistic assessment of practitioner perceptions, and additional opportunities for refinement prior to conducting the second, more controlled evaluation.

[21] https://github.com/arpitthool/SafeAIMerge

Participants

Participants were recruited using a multi-channel strategy that combined targeted outreach through professional networks, direct invitations to industry practitioners, and public calls shared via Linkedin[22], along with the inclusion of technically experienced graduate students from Virginia Tech with relevant software development backgrounds. The 46 survey participants represented a range of professional roles, including software engineers, DevOps engineers, security analysts, and technical leads. They were employed at organizations such as Adobe,[23] Amazon,[24] IBM,[25] Microsoft,[26] and several other companies. On average, participants reported 4.8 years of technical work experience (range: 0–20 years), with 32 respondents having up to five years of experience and 14 reporting more than five years.

Survey Results

Quantitative Results Table 7.1 summarizes the central tendency and inferential results for the five core likert metrics: user-friendliness, integration ease, PR usefulness, adoption likelihood, and overall satisfaction. Across all measures, respondents gave high ratings: median scores were between 4 and 5, and means ranged from 4.35 to 4.85. Wilcoxon signed-rank tests confirmed that scores were significantly above the neutral midpoint of 3 ($p < 0.001$ for all items), with medium-to-large effect sizes ($r = 0.37$ — 0.55). Recommendation rates were unanimous: 100% of respondents indicated they would recommend the tool to colleagues. For integration ease, 38 participants reported that the tool "fits in seamlessly" into CI/CD pipelines, while 8 indicated "some adjustments needed." PR usefulness was strongly endorsed, with 36 participants "strongly agreeing" and 10 "agreeing" that embedding reports

[22] https://www.linkedin.com/
[23] https://www.adobe.com/
[24] https://amazon.com/
[25] https://www.ibm.com/us-en
[26] https://www.microsoft.com/

Table 7.1: Median, mean, Wilcoxon signed-rank test results, and effect sizes (r) for survey responses

Measure	**Median**	**Mean**	p	r
User-friendliness	5	4.78	$< .001$	0.55
Integration ease	5	4.74	$< .001$	0.47
PR usefulness	5	4.78	$< .001$	0.46
Adoption likelihood	4	4.35	$< .001$	0.37
Satisfaction	5	4.85	$< .001$	0.43

in GitHub PRs was valuable.

Qualitative Results Analysis of open-ended responses highlighted several recurring themes. Participants valued the *Clarity of LLM-generated Summaries*, which reduced the effort needed to interpret raw security alerts. Others emphasized the importance of *Workflow Compatibility*, noting that the GitHub PR integration minimized disruption. Common additional suggestions for improvement included enhancements to the *Reporting Format*, such as more visual security report and PR comment. Overall, the results indicated that practitioners perceived *SafeAIMerge* as a user-friendly, lightweight, and a valuable addition to Agile workflows. High satisfaction scores, unanimous recommendation rates, and strong endorsements of GitHub PR integration underscored the tool's potential for adoption in practice.

Feedback-Driven Refinement

While survey results indicated that *SafeAIMerge* was already perceived as highly usable, lightweight, and well aligned with existing GitHub CI/CD workflows, participants also provided constructive suggestions for further improving *SafeAIMerge*. These comments were not indicative of fundamental usability issues, but rather reflected opportunities for incre-

mental refinement. Based on this feedback we made further refinements to *SafeAIMerge* which focused on improving the visual presentation of security feedback while preserving existing workflows. We redesigned the generated security report from a plain text format to a structured HTML report that supports interactive elements. The revised report organized findings into three clearly delineated sections: (i) new alerts introduced by the PR, (ii) alerts resolved by the PR, and (iii) pre-existing alerts already present on the `main` branch. Each section is implemented as an expandable dropdown, allowing developers to progressively disclose details only when needed. This design enables quick high-level assessment of security impact while still providing access to full alert metadata, supporting both rapid triage and deeper investigation. Similarly, we refined the GitHub PR comment generated by *SafeAIMerge*. Instead of a single plain text comment, the updated PR feedback presents three expandable sections corresponding to new, resolved, and existing alerts. Each dropdown includes a concise summary of its respective category, allowing reviewers to immediately understand how the PR affects security without scrolling through verbose outputs. This structured presentation was designed to better align with code review practices, where reviewers often seek concise, high-signal information that can be expanded on demand. Collectively, these refinements reflect an iterative, feedback-driven design approach that prioritizes usability and developer experience alongside technical accuracy. This approach aligns with previous work (Chapter 4 and Chapter 6) showing that security tool adoption in Agile and CI/CD settings depends not only on automation, but also on clear, actionable, and workflow-compatible feedback.

7.4.2 Phase II: Controlled User Study

While the online survey provided early insight into perceived usability and adoption potential, it did not capture how developers actually perform security-related tasks when using

the tool. To address this limitation and to explicitly answer the research questions (Section 7.1) we conducted a controlled, task-based user study comparing *SafeAIMerge* against the baseline ZAP workflow.

To answer RQ1, we measured participants' perceived workload during vulnerability remediation using NASA-TLX questionnaire, a widely adopted instrument for assessing subjective workload in human–computer interaction studies [249]. To answer RQ2, we evaluated remediation effectiveness and efficiency using objective task outcomes, including the number of vulnerabilities successfully resolved and the time required to resolve at least one vulnerability within the allotted task duration. To answer RQ3, we assessed developers' workflow preferences using a comparative survey that captured perceived ease of use, clarity and actionability of security reports, perceived time required to interpret vulnerabilities, and overall workflow preference after participants experienced both conditions.

Study Design

The controlled user study followed a within-subjects design with two conditions:

1. **Baseline workflow:** participants used a standard ZAP integration in GitHub Actions CI/CD pipeline and reviewed the resulting ZAP security report to identify and remediate the reported vulnerabilities.

2. ***SafeAIMerge* workflow:** participants used *SafeAIMerge* integration in GitHub Actions CI/CD pipeline and reviewed the resulting security report, which included LLM-generated summaries and recommended remediation steps presented within the PR context.

Each participant completed both conditions in a single session lasting approximately 45—

60 minutes. At the beginning of each session, the researcher provided an overview of the study goals and procedures. Participants were assured that their identity and responses would remain confidential and anonymized , consistent with the study invitation and consent documentation. Then they received a brief introduction to DAST, including what a DAST scan does and how it can be integrated into GitHub Actions CI/CD pipelines. Then they completed a short demographic questionnaire capturing basic background information (e.g., organization, role, and CI/CD experience).

In the baseline condition, the researcher introduced a simple GitHub repository[27] that consisted of a simple JavaScript web application. Participants along with the researcher integrated the standard ZAP scans into the repository's GitHub Actions workflow and then they triggered a prepared vulnerable PR that in-turn triggered the ZAP scan. After the workflow completed, participants were asked to review the generated ZAP security report,[28] identify the vulnerabilities, and attempt to fix as many vulnerabilities as possible within the allotted task duration (15 minutes). Immediately after completing the baseline task (or reaching the time limit), the participants viewed the results (if they pushed any code to the PR) and then completed a NASA-TLX workload questionnaire consisting of the standard six workload dimensions. Similarly, in the *SafeAIMerge* condition, participants moved to a second repository[29] which was another simple JavaScript web application. Participants along with the researcher integrated the *SafeAIMerge* into GitHub Actions workflow, created a prepared PR to trigger the automated scans. Then the participants were asked to review the *SafeAIMerge* security report.[30] In contrast to the baseline workflow, *SafeAIMerge* provided summarized security alerts, developer-oriented remediation guidance, and an enhanced report experience within the PR context. Participants again attempted to fix as

[27] https://github.com/arpitthool/maze-escape
[28] Baseline ZAP security report
[29] https://github.com/arpitthool/pac-man
[30] *SafeAIMerge* security report

many vulnerabilities as possible within the allotted time (15 minutes). After completing this task (or reaching the time limit), the participants viewed the results (if they pushed any code to the PR) and then completed the NASA-TLX workload questionnaire a second time.

To ensure a fair and controlled comparison between workflows, both repositories used in the study were configured to produce security reports with the same number, severity distribution, and types of vulnerabilities. The prepared PRs in both the baseline ZAP and *SafeAIMerge* conditions triggered identical ZAP scan configurations and resulted in security reports containing the same set of alerts. This design choice ensured that any observed differences in workload, task completion time, or vulnerability resolution could be attributed to differences in how security feedback was presented and integrated into the workflow, rather than differences in the underlying security findings. Across both conditions, the generated security reports included a mix of medium-, low-, and informational-severity findings. Specifically, each report contained two medium-severity alert types—*Content Security Policy (CSP) Header Not Set*[31] and *Missing Anti-clickjacking Header*[32]—with four instances of each. In addition, several low-severity issues were reported, including *Insufficient Site Isolation Against Spectre Vulnerability*[33], *Permissions Policy Header Not Set*[34], *Server Version Information Leakage via HTTP Headers*[35], and *Missing X-Content-Type-Options Header*[36]. A small number of informational alerts related to cacheable or storable content were also present. These issues were selected intentionally, as they require participants to understand the reported vulnerability, locate the relevant configuration or middleware, and apply a concrete fix—making them representative of common, real-world web security remediation tasks.

[31] Content Security Policy (CSP) Header Not Set
[32] Missing Anti-Alickjacking Header
[33] Insufficient Site Isolation Against Spectre Vulnerability
[34] Permissions Policy Header Not Set
[35] Server Version Information Leakage via HTTP Headers
[36] X-Content-Type-Options Header Missing

To answer RQ1 on perceived workload, we used a modified version of the NASA-TLX. Participants rated their experience across six workload dimensions using five-point Likert-style scales, these included:

- **Mental Demand**: how mentally demanding the task was (Very Low to Very High).
- **Physical Demand**: how physically demanding the task was (Very Low to Very High).
- **Temporal Demand**: how hurried or rushed the task felt (Very Low to Very High).
- **Perceived Performance**: how successful participants felt in accomplishing the task (Highly Unsuccessful to Highly Successful).
- **Effort**: how hard participants had to work to achieve their level of performance (Very Low to Very High).
- **Frustration**: the extent to which participants felt insecure, discouraged, irritated, stressed, or annoyed during the task (Very Less Frustrated to Very Highly Frustrated).

All NASA-TLX items were administered after each task condition to capture condition-specific workload perceptions. Lower scores on mental demand, temporal demand, effort, and frustration indicate lower perceived workload, while higher scores on perceived performance indicate greater task success.

To answer RQ2, in addition to the NASA-TLX responses, we recorded task performance, measured by the number of vulnerabilities successfully resolved and task completion time, defined as the time required to resolve at least one vulnerability. Participants who did not resolve any vulnerability within the allotted duration were assigned the maximum task time.

And to answer RQ3, after completing both tasks, participants completed a short comparative survey designed to elicit direct preferences between the two workflows. The survey

consisted of five forced-choice questions, each asking participants to compare the baseline ZAP workflow with the *SafeAIMerge* workflow along a specific dimension. For each question, participants selected one of four response options: (i) Baseline ZAP workflow, (ii) *SafeAIMerge* workflow, (iii) Both equally, or (iv) Neither.

The comparative questions focused on the following dimensions:

- **Overall ease of use of the workflow**, to assess the general usability of each workflow.
- **Effectiveness in helping participants understand the reported vulnerability and remediation steps**, to evaluate whether LLM-assisted, contextualized feedback improves comprehension and ability to act on security findings.
- **Overall workflow preference**, to capture participants' holistic judgment after experiencing both conditions.
- **Clarity and actionability of the generated security reports**, to isolate the impact of report presentation and guidance quality.
- **Perceived time required to interpret vulnerabilities**, to assess whether differences in feedback presentation affected participants' sense of efficiency and time pressure, complementing the objective task completion time measurements.

To mitigate learning and ordering effects, the study was counterbalanced: 6 out of 12 participants completed the baseline condition first, while the other 6 completed the *SafeAIMerge* condition first. The user study questions are provided in the Appendix C.2.

Participants

A total of 12 participants took part in the study, representing a range of technical roles and organizational contexts. Participants included graduate students and software engineers. Most participants were computer science graduate students from Virginia Tech, while others were employed in industry across organizations such as CNH Capital[37], Torc Robotics[38]. Participants reported an average of 4.25 years of technical work experience (range: 0–15 years). Specifically, two participants had 0–1 years of experience, four reported 2–3 years, five reported 4–5 years, and one participant reported 15 years of professional experience. With respect to development practices, 11 of the 12 participants reported prior experience working with CI/CD pipelines, but none of the participants had prior experience with security-focused workflows, allowing the study to examine how developers without specialized security backgrounds engage with and respond to security feedback. This participant profile is generally representative of industry context, where CI/CD pipelines are widely used but developers often lack formal experience with security-focused workflows [250].

Results

RQ1: Developer Workload To answer RQ1, we analyzed participants' responses to the NASA-TLX questions to compare the baseline ZAP workflow with the *SafeAIMerge* workflow. For each participant and condition, an overall (raw) NASA-TLX workload score was computed by averaging the six workload dimensions after mapping Likert-style responses to a 0—100 scale and inverting the "perceived performance" dimension so that lower values indicate lower workload. Condition-level descriptive statistics (mean, median, and standard deviation) were then calculated across participants. Because the study followed a within-

[37] https://www.cnhcapital.com/
[38] https://torc.ai/

Table 7.2: NASA-TLX workload dimensions: baseline ZAP vs. *SafeAIMerge*.

Dimension	**Baseline Mean**	**Tool Mean**	**Median Diff**	W	p	r
Mental Demand	72.92	20.83	-50.00	0.0	0.003	0.77
Physical Demand	25.00	12.50	-12.50	2.5	0.084	0.39
Temporal Demand	39.58	16.67	-25.00	0.0	0.008	0.69
Performance (inv.)	77.08	14.58	-62.50	0.0	0.003	0.79
Effort	66.67	16.67	-50.00	0.0	0.005	0.75
Frustration	54.17	10.42	-43.75	0.0	0.007	0.71

Performance scores were inverted so that lower values indicate lower workload.

subjects design and the data did not satisfy normality assumptions, we used the Wilcoxon signed-rank test for paired comparisons. In addition to statistical significance, we report effect sizes using the rank-biserial correlation (r), calculated as $r = Z/\sqrt{N}$, where Z is the standardized Wilcoxon test statistic and N is the number of paired observations.

Table 7.2 reports results for each NASA-TLX dimension. Participants experienced significantly lower mental demand, temporal demand, effort, frustration, and performance-related workload when using *SafeAIMerge*. Physical demand showed a downward trend but did not reach statistical significance. Across all significant dimensions, effect sizes were large, indicating that the observed reductions in workload were both statistically and practically meaningful.

Table 7.3 summarizes the overall workload results. Participants reported substantially lower overall workload when using *SafeAIMerge* compared to the baseline ZAP workflow. The median raw NASA-TLX score decreased from 58.3 in the baseline condition to 12.5 in the *SafeAIMerge* condition. This reduction was statistically significant ($W = 1.0$, $p < 0.001$) and associated with a large effect size ($r = 0.85$). These results suggest that integrating summarized, PR-contextual security feedback directly into developer workflows can substantially reduce cognitive and emotional workload during security remediation tasks.

Table 7.3: Overall NASA-TLX workload comparison between baseline ZAP and *SafeAIMerge*.

Condition	**Median**	**Mean**	**Standard Deviation**	**Wilcoxon W / p**
Baseline ZAP	58.33	55.90	22.51	$W = 1.0$, $p < 0.001$
SafeAIMerge	12.50	15.28	12.09	

$r = 0.85$ (large effect). Lower scores indicate lower perceived workload.

RQ2: Remediation effectiveness and efficiency To answer RQ2, we examined task completion time and successful vulnerability resolution to assess how each workflow supported practical security remediation. Task completion time was defined as the time required to successfully resolve at least one vulnerability. Participants who did not resolve any vulnerability within the allotted duration were assigned the maximum task time of 15 minutes.

For the baseline ZAP workflow, all participants reached the maximum allotted time of 15 minutes, resulting in both a mean and median completion time of 15 minutes. Only one participant successfully resolved a vulnerability within the time limit when using the baseline workflow. In contrast, participants using the *SafeAIMerge* workflow completed remediation tasks substantially faster. Completion times for the *SafeAIMerge* condition ranged from 2 to 15 minutes, with a mean completion time of 5.9 minutes and a median of 5 minutes. Because baseline task times were uniformly capped at the upper time limit, a paired statistical comparison of completion times was not meaningful. Instead, these descriptive statistics highlight a substantial and consistent reduction in time-to-action when participants used *SafeAIMerge*. In terms of task performance, 11 out of 12 participants successfully resolved at least one security issue when using the *SafeAIMerge* workflow, compared to only one participant in the baseline ZAP condition. The most commonly resolved issue was a *Missing*

Anti-Alickjacking Header[39], which was classified as a medium-level severity. It is to be noted that this alert would appear at the top in the security report in both conditions, as the alerts were sorted by the severity levels. A few ($n = 2$) participants also resolved other security issues, such as *X-Content-Type-Options Header Missing*[40] and *Content Security Policy (CSP) Header Not Set*[41].

Participants using *SafeAIMerge* were often able to directly follow the summarized vulnerability descriptions and remediation guidance provided within the PR. In many cases, participants transitioned quickly from reading the report to implementing the fix with minimal trial-and-error. In contrast, during the baseline ZAP condition, participants frequently spent a large portion of the task duration interpreting the raw scan output, visiting various reference links and attempting to determine how reported findings mapped to concrete code changes. These results demonstrate that *SafeAIMerge* improves both the effectiveness and efficiency with which developers translate security findings into concrete fixes.

RQ3: Workflow preference To answer RQ3, we analyzed responses to the comparative survey completed after participants experienced both workflows. Responses were unanimous across all survey questions. All 12 participants reported the *SafeAIMerge* workflow as *Easier to use*, *More effective in helping them understand reported vulnerabilities and associated remediation strategies*, *Overall preference*, *Having clearer and more actionable reports* and one which *Required less time to interpret vulnerabilities*. Taken together, the comparative survey results provide strong evidence that developers preferred the *SafeAIMerge* workflow over the baseline ZAP workflow when addressing security vulnerabilities in CI/CD pipelines. When combined with the workload and task performance findings, these results indicate that

[39] Missing Anti-Alickjacking Header
[40] X-Content-Type-Options Header Missing
[41] Content Security Policy (CSP) Header Not Set

SafeAIMerge not only reduces perceived effort but is also favored by developers as a more usable and supportive security workflow.

7.5 Limitations

This study was subjected to several limitations that should be considered when interpreting the results, particularly with respect to internal, external, and construct and conclusion validity.

Internal validity All participants completed both task conditions within a single study session. Although the study was counterbalanced to mitigate ordering and learning effects, some familiarity with the task domain, development environment, or vulnerability types may have carried over between conditions and influenced participant behavior. In addition, subjective measures such as NASA-TLX ratings and comparative workflow preferences may be influenced by participant expectations or social desirability effects. The presence of a researcher during the study sessions may also have affected how participants approached the tasks. To reduce these risks, standardized task instructions were used, and participants were assured that their responses would remain anonymized.

External validity The external validity of the findings is constrained by the scope and setting of the study. The evaluation was conducted using two intentionally constructed JavaScript web applications with a controlled and identical set of security vulnerabilities across conditions. While this design ensured parity between workflows and enabled a fair comparison, real-world software systems often involve larger codebases, more complex architectures, and a broader range of vulnerability types. In addition, the study focused on a

single DAST tool (ZAP) and a specific CI/CD setup based on GitHub Actions, which may limit the generalizability of the results to other tools, development methodologies, CI/CD infrastructures, or organizational contexts.

Construct and conclusion validity Construct and conclusion validity are influenced by how workload, performance, and task success were operationalized. The evaluation focused primarily on short-term task performance, subjective workload, and immediate workflow preferences. While these measures are well-established in human–computer interaction research, they do not capture longer-term outcomes such as sustained developer productivity, improvements in software security posture, or organizational adoption. In addition, task completion time in the baseline condition was uniformly capped at the maximum allotted duration, which limited the use of inferential statistical tests for time-based comparisons. As a result, time-related findings are reported descriptively rather than through formal hypothesis testing.

7.6 Future Work

The findings of this study open several promising directions for future research on integrating LLM-assisted security tools into developer workflows. A natural next step is to conduct larger-scale evaluations involving more participants across diverse organizational roles, experience levels, and development contexts. Such studies would help assess the generalizability of the observed workload reductions, performance improvements, and workflow preferences beyond the controlled setting used in this work. Future studies should also evaluate *SafeAIMerge* in production-scale repositories with more complex architectures and a broader range of vulnerability types. While the current study focused on intentionally con-

structed applications with controlled alert parity, real-world systems often generate larger, noisier security reports and involve deeper dependencies across services and configurations. Examining how summarized, PR-contextual security feedback scales to these environments is an potential avenue for continued investigation.

Longitudinal and field-based studies represent another critical direction for future work. Observing developers' interactions with *SafeAIMerge* over extended development cycles could provide insight into learning effects, sustained usability, and long-term adoption. Such studies would also enable evaluation of outcomes that were beyond the scope of the present work, including changes in developer behavior, collaboration patterns, and organizational security practices. Future work should additionally explore alternative study designs and performance metrics. Employing longer or adaptive task time limits may allow for more granular analysis of time-to-remediation differences, while integrating objective security metrics—such as vulnerability recurrence, fix correctness, or downstream security impact—could complement subjective workload measures. Combining controlled experiments with quantitative analyses of security outcomes would strengthen conclusions about the effectiveness of LLM-assisted security tooling.

Finally, the approach presented in this paper could be extended beyond DAST and web application security. Investigating how similar LLM-based summarization and contextualization techniques apply to other security activities, such as SAST, dependency scanning, or infrastructure-as-code security, may further broaden the applicability of this work. Together, these directions offer a path toward more human-centered, context-aware security tooling that better aligns with modern Agile and CI/CD development practices.

7.7 Conclusion

This chapter presented *SafeAIMerge*, an LLM-assisted, CI/CD-integrated approach for delivering contextualized and developer-oriented security feedback during code reviews. By combining DAST with PR-aware summarization and remediation guidance, *SafeAIMerge* aims to bridge the gap between security tooling and everyday developer workflows. We evaluated *SafeAIMerge* through a two-phase mixed-methods study. In Phase I, an online survey provided evidence that *SafeAIMerge* was perceived as highly usable, well integrated into CI/CD workflows, and valuable for PR-based security feedback, with unanimous recommendation and strong adoption potential among practitioners. These findings motivated a deeper task-based evaluation and informed the design of the tool's reporting and integration features. In Phase II, we conducted a controlled within-subjects user study comparing *SafeAIMerge* against the baseline ZAP workflow using identical security findings across conditions. The results showed that presenting security feedback in a summarized, PR-contextual form substantially reduced perceived workload, accelerated vulnerability remediation, and improved task success. Participants consistently resolved more security issues in less time, reported significantly lower cognitive and emotional workload, and unanimously preferred the *SafeAIMerge* workflow over the baseline approach.

Beyond performance gains, our results highlight the importance of human-centered design in security tooling. Across both phases, participant feedback and researcher observations suggest that actionable summaries, clear remediation guidance, and tight integration into CI/CD and code review contexts enable developers to move more efficiently from vulnerability awareness to concrete fixes, reducing cognitive and emotional load that can otherwise hinder security practice adoption. While the evaluation was conducted in a controlled setting, this work contributes both a practical tool and systematic evidence that LLM-enhanced,

context-aware security reporting can improve developer experience and effectiveness during security remediation tasks. These findings encourage further exploration of human-centered, AI-assisted approaches to integrating security practices into modern Agile and CI/CD-driven software development.

Chapter 8

Conclusion

By focusing on developer perceptions, this dissertation investigated how the DAST security practice can be effectively integrated into Agile software development. Across multiple empirical studies and tool-building efforts, the dissertation demonstrated that practitioners perceive DAST can be effectively integrated into Agile when (i) practitioner constraints and perceptions are explicitly measured, (ii) complex tool outputs are transformed into actionable information, and (iii) security feedback is embedded directly into developers' existing workflows and CI/CD pipelines. These conditions directly address the core challenges identified in Chapter 1—process integration, tool complexity, and practical implementation.

The central claim supported by this dissertation is that security practices—particularly DAST can be efficiently integrated into Agile development through LLM-enhanced security tooling and practitioner-centered implementation strategies. In the remainder of this chapter, we summarize this dissertation's main contributions, discuss their implications for researchers and practitioners, outline key limitations, and present directions for future work.

8.1 Summary of Contributions

The dissertation contributes to secure Agile software engineering through a set of complementary studies that collectively address three recurring challenges identified in the introduction (Chapter 1): integrating DAST into Agile processes, reducing the complexity of DAST tool

outputs, and providing empirically grounded guidance for implementation.

8.1.1 Practitioner Perspectives on Security Activities in Agile Development

Firstly, Chapter 4 provided an empirical evidence on how Agile practitioners perceive and experience security activities in real development environments. Through a survey study with software practitioners, we examined (i) which security activities are adopted in Agile teams, (ii) how practitioners evaluate their effectiveness and their willingness to adopt them, and (iii) how these activities impact productivity, confidence, and day-to-day work. The findings indicated that practitioners generally recognize the value of security activities and do not perceive them as fundamentally incompatible with Agile when they are aligned with iterative workflows. At the same time, the results highlighted that adoption is uneven, and that automation and high-quality feedback are critical conditions for security activities to remain viable under Agile time constraints. These findings established a practitioner-grounded foundation for the subsequent contributions of the dissertation by clarifying what Agile teams will actually adopt, what they struggle with, and what kinds of interventions are likely to reduce friction.

8.1.2 LLM Summarization for Human-Centric DAST Reporting

In Chapter 5 we addressed the problem that DAST tools often produce reports that are verbose, difficult to interpret, and costly to act upon—particularly for developers without deep security expertise. We investigated the use of LLMs to summarize DAST alerts and improve their clarity and comprehensibility for practitioners. Through an empirical evaluation with software practitioners, we found strong evidence that LLM-generated summaries

can substantially improve the perceived accessibility of DAST outputs and are frequently preferred over traditional alert formats. This contribution established that LLMs can play a practical role in improving the usability of security tooling by translating dense security documentation into concise, developer-oriented explanations and remediation guidance—thereby reducing the cognitive burden that often prevents security findings from being addressed promptly.

8.1.3 Case Study of Real-World DAST Integration in an Agile Team

Next, in Chapter 6 we provided an in-place understanding of what it takes to integrate DAST into an operational Agile team and CI/CD pipeline. Through an action-research case study, we documented adoption challenges and mitigation strategies observed during real deployment and use. The results highlighted that the success of DAST integration in Agile depends not only on tool availability, but also on workflow alignment, prioritization support, pipeline robustness, and the distribution of security expertise within the team. Importantly, the study showed that DAST can be adopted with minimal disruption when introduced incrementally, embedded into existing routines, and supported by automation that reduces manual coordination and report interpretation effort. This contribution strengthens the dissertation's overall argument by connecting practitioner perceptions to real-world implementation constraints and demonstrating how security practices become sustainable when integrated as an iterative engineering activity rather than an external gate.

8.1.4 *SafeAIMerge*: CI/CD-Integrated DAST and LLM Feedback in Pull Requests

Finally in Chapter 7 we operationalized the preceding findings by designing and evaluating *SafeAIMerge*, a tool that integrates DAST execution and LLM-based summarization and remediation directly into GitHub PR workflows. *SafeAIMerge* is motivated by a practical observation: even when security scanning exists, developers often disengage when results require them to interpret long and complex reports in external interfaces. By embedding security feedback into PR context, *SafeAIMerge* presents actionable summaries and reduces context switching at the point where developers are already making changes.

We evaluated *SafeAIMerge* through both a formative practitioner survey (capturing perceived usefulness and usability) and a controlled within-subjects summative user study (capturing comparative workload and effectiveness outcomes against a baseline DAST workflow). The combined results provide evidence that PR-integrated, LLM-assisted security feedback can reduce cognitive and emotional workload, improve vulnerability remediation effectiveness, and increase practitioner preference for security-involved workflows. This contribution serves as the dissertation's most direct demonstration that security integration can be simultaneously (i) workflow-compatible, (ii) developer-centered, and (iii) empirically validated.

8.2 Implications

8.2.1 Implications for Practitioners

This dissertation provides several actionable implications for teams aiming to strengthen security in Agile environments, grounded in empirical findings. First, teams should treat

security integration as a workflow design problem rather than only a tooling problem. Across the practitioner survey (Chapter 4) and the action-research case study (Chapter 6), security activities were more likely to be adopted and sustained when they aligned with iterative development rhythms and minimized context switching, rather than being introduced as standalone or external processes. Second, automation is necessary but not sufficient. Both the case study (Chapter 6) and the *SafeAIMerge* evaluations (Chapter 7) show us that automated scanning must be paired with feed back that is concise, contextualized, and actionable; otherwise, teams risk alert fatigue and low engagement even when tools are deployed. Third, LLMs are a promising mechanism for improving the usability of security tooling, especially for DAST. The empirical results from the LLM summarization study (Chapter 5) and the *SafeAIMerge* evaluations (Chapter 7) demonstrate that LLM-generated summaries significantly improved perceived clarity and reduced cognitive burden, increasing the likelihood that vulnerabilities were addressed within development cycles. Finally, integrating security feedback into PRs and CI/CD systems can convert security from an external auditing activity into a normal part of code review and change management. This approach is especially valuable in Agile teams where time and attention are constrained and where security work must compete with rapid delivery of features.

8.2.2 Implications for Researchers

For researchers, this dissertation provides empirical grounding for studying secure Agile development as a socio-technical problem. Across all studies, the primary obstacles to effective security integration are not limited to detection capability; they often lie in communication, workflow fit, and the human effort required to interpret and act on findings. These observations are consistent with the literature survey findings 2.4, which shows that much of the existing research on security analysis tools and LLM-based techniques prioritizes tech-

nical accuracy, while offering comparatively limited insight into developer insight/workflow integration.

The work (Chapter 7) demonstrates a viable evaluation strategy for developer-centered security tooling that combines perception-based measures (e.g., preferences and perceived usefulness) with task-based evidence (e.g., remediation success and workload). The mixed-methods approach, applied across multiple studies in this dissertation, provides a more comprehensive understanding of both how developers experience security tools and how those tools affect actual remediation performance. Finally, this work adds to the emerging body of research on LLMs in software engineering by showing that LLMs can meaningfully improve the accessibility of security outputs when used as a translation and summarization layer tightly integrated into developer workflows.

8.3 Limitations

This dissertation has several limitations that motivate additional research. First, several studies rely on practitioner self-reporting (e.g., survey-based measures of perceptions, willingness, and perceived impacts), which can be influenced by individual experience, organizational context, and recall bias. Second, the studies focus primarily on DAST and web application contexts. While DAST is widely used and well aligned with the dissertation's goals, generalization to other security activities (e.g., SAST, dependency scanning, infrastructure security) requires additional investigation. Third, the LLM-based components introduce known risks, including potential hallucination, omission of critical details, and variability across models and prompts. Although this dissertation evaluates perceived clarity and usefulness and demonstrates workflow-level benefits, future work should expand validation to include rigorous correctness and safety assessments of generated guidance. Importantly, in

this dissertation LLMs function as decision-support mechanisms rather than autonomous security agents, and are evaluated in terms of their impact on developer understanding and workflow effectiveness. Finally, the case study and tool evaluations reflect specific environments, workflows, and task designs. While the dissertation uses multiple complementary studies to strengthen external validity, broader replication across organizations and domains would increase generalizability.

8.4 Future Work

This dissertation opens several directions for future research and engineering. First, future work should expand from summarization to *end-to-end security assistance* in Agile workflows, including interactive explanation, prioritization support, architecture constraints, and organizational policies. Second, future work should incorporate *accuracy- and safety-oriented evaluation* of LLM-generated security guidance. This includes measuring factual correctness, detecting unsafe recommendations, and designing guardrails (e.g., retrieval augmentation, structured outputs, or verifier models) that preserve usefulness while reducing risk. Third, an important direction is *multi-activity security integration* that studies how DAST interacts with other security activities in CI/CD pipelines (e.g., SAST, secret scanning, dependency vulnerability analysis). Understanding how combined tool feedback affects developer workload, attention, and remediation choices would provide a more complete picture of secure Agile development in practice. Fourth, future work should explore *personalization and role-specific security feedback*. Different stakeholders (developers, testers, managers, security engineers) need different levels of abstraction and different decision support. LLM-based reporting systems could be adapted to produce multiple views of the same finding tailored to stakeholder goals. Finally, future work should pursue *longitudinal, in-the-wild deployment*

studies of workflow-integrated security tooling. Measuring sustained adoption, changes in remediation behavior over time, and downstream security outcomes would provide stronger evidence of real-world impact.

Appendices

Appendix A

Security Practices in Agile Software Development

A.1 Online Survey Questionnaire

1a. Name: _____

1b. Email: _____

1c. Current Company/Organization: _____

1d. Current Role: _____

1e. Total years of technical work experience: _____

2. In your opinion, how important is software security for your team?)

- ☐ Not at all important
- ☐ Slightly important
- ☐ Moderately important
- ☐ Very important
- ☐ Extremely important

3. Do you use Agile software development methodology in your organization?

□ Yes □ No

4. How much do you agree with the statement: "Software developed through Agile methods are relatively less secure when compared to software developed through sequential software development life-cycle processes like Waterfall"?

□ Strongly Disagree

□ Disagree

□ Neither agree nor disagree

□ Agree

□ Strongly agree

5. Does your team include any security activities in the Agile process?

□ Yes □ No

6a. What are these security practices used in your Agile process? (*Optional*)

_____

_____

6b. What is your take on these security practices used in your team? (*Optional*)

_____

_____

6c. How has the inclusion of these security practices affected the team productivity? (*Optional*)

_____

_____

6d. How has the involvement of these security practices affected the software product? (*Optional*)

_____

_____

6e. How has the inclusion of these security practices affected the organization? (*Optional*)

_____

_____

6f. How has the involvement of these security practices affected your day-to-day activities?(*Optional*)

_____

_____

6g. How was the sprint velocity affected? (*Optional*)

_____

_____

7. After using these security practices are you more confident in the security of the software you are building?

□ Not confident at all

□ Slightly confident

□ Somewhat confident

□ Fairly confident

□ Completely confident

How effective would each security practice be in increasing the security and robustness of the software, if your team would include it in the Agile software development process?

8a. Addressing security from early iterations with requirements and testing.

- ☐ Not at all effective
- ☐ Slightly effective
- ☐ Moderately effective
- ☐ Very effective
- ☐ Extremely effective

8b. Clearly stating security requirements, that are expected to be in the production software.

- ☐ Not at all effective
- ☐ Slightly effective
- ☐ Moderately effective
- ☐ Very effective
- ☐ Extremely effective

8c. Adding a Security specialist or Security master to your team.

- ☐ Not at all effective
- ☐ Slightly effective
- ☐ Moderately effective
- ☐ Very effective
- ☐ Extremely effective

8d. Assigning additional points or weight to a ticket, considering the level of impact the ticket will have on software security.

- ☐ Not at all effective
- ☐ Slightly effective
- ☐ Moderately effective
- ☐ Very effective
- ☐ Extremely effective

8e. Iterative and incremental vulnerability and penetration testing.

- ☐ Not at all effective
- ☐ Slightly effective
- ☐ Moderately effective
- ☐ Very effective
- ☐ Extremely effective

8f. Iterative and incremental security static analysis.

- ☐ Not at all effective
- ☐ Slightly effective
- ☐ Moderately effective
- ☐ Very effective
- ☐ Extremely effective

8g. Iterative and incremental risk analysis, countermeasure graphs.

- ☐ Not at all effective
- ☐ Slightly effective
- ☐ Moderately effective
- ☐ Very effective
- ☐ Extremely effective

8h. Automatic testing adding vulnerabilities analysis, risk assessment into the deployment

pipeline

- ☐ Not at all effective
- ☐ Slightly effective
- ☐ Moderately effective
- ☐ Very effective
- ☐ Extremely effective

How willing are you to include each security practice in your Agile software development process?

9a. Addressing security from early iterations with requirements and testing

- ☐ Not at all willing
- ☐ Slightly willing
- ☐ Moderately willing
- ☐ Very willing
- ☐ Extremely willing

9b. Clearly stating security requirements, that are expected to be in the production software.

- ☐ Not at all willing
- ☐ Slightly willing
- ☐ Moderately willing
- ☐ Very willing
- ☐ Extremely willing

9c. Adding a Security specialist or Security master to your team.

- ☐ Not at all willing

□ Slightly willing

□ Moderately willing

□ Very willing

□ Extremely willing

9d. Assigning additional points or weight to a ticket, considering the level of impact the ticket will have on software security.

□ Not at all willing

□ Slightly willing

□ Moderately willing

□ Very willing

□ Extremely willing

9e. Iterative and incremental vulnerability and penetration testing.

□ Not at all willing

□ Slightly willing

□ Moderately willing

□ Very willing

□ Extremely willing

9f. Iterative and incremental security static analysis.

□ Not at all willing

□ Slightly willing

□ Moderately willing

□ Very willing

□ Extremely willing

9g. Iterative and incremental risk analysis, countermeasure graphs.

□ Not at all willing

□ Slightly willing

□ Moderately willing

□ Very willing

□ Extremely willing

9h. Automatic testing adding vulnerabilities analysis, risk assessment into the deployment pipeline

□ Not at all willing

□ Slightly willing

□ Moderately willing

□ Very willing

□ Extremely willing

Appendix B

LLM Summarization for Human-Centric DAST Reports

B.1 Online Survey Questionnaire

Background Questions

1a. Name: _____

1b. Email: _____

1c. Current Company/Organization: _____

1d. Current Role: _____

1e. Total years of technical work experience: _____

2a. How familiar are you with security reports of software products?

- ☐ Very Unfamiliar
- ☐ Somewhat Unfamiliar
- ☐ Neutral

□ Somewhat Familiar

□ Very Familiar

2b. How comfortable are you with knowing and understanding security issues mentioned in security reports?

□ Very Uncomfortable

□ Somewhat Uncomfortable

□ Neutral

□ Somewhat Comfortable

□ Very Comfortable

3a. Have you used automated security tools for scanning and identifying issues in software projects? □ Yes □ No

3b. If yes, please tell us which tools you used and briefly describe your experience with these tools. (*Optional*)

_____

_____

3c. If yes, how often do you use these tools? (*Optional*)

_____

_____

4a. How would you rate the usefulness of traditional DAST security reports in addressing and resolving security issues?

□ Very Useless

□ Somewhat Useless

☐ Neutral

☐ Somewhat Useful

☐ Very Useful

4b. What challenges do you face when dealing with traditional DAST security reports? (*Optional*)

Please review the security report provided:

The summary reports that a vulnerable JavaScript library called jQuery was detected with the version 1.8.1.min. The vulnerabilities found on this library include CVE-2012-6708, CVE-2015-9251, and CVE-2015-9251. Additionally, the library may have introduced DOM-based vulnerabilities that can be exploited to hijack user accounts. The use of third-party JavaScript libraries can introduce a range of vulnerabilities, including some that can be used to hijack user accounts. Common JavaScript libraries typically enjoy the benefit of being heavily audited, and it's recommended to apply security updates promptly if available. Finally, it's important to identify which library vulnerabilities apply to your website and prioritize applying all available security updates.

5a. Rate its clarity:

☐ Very Unlear

☐ Somewhat Unlear

☐ Neutral

☐ Somewhat Clear

☐ Very Clear

5b. Were you able to easily understand the security issues? ☐ Yes ☐ No

Please review the security report provided:

The article discusses a security vulnerability in the JavaScript library jQuery, specifically versions 1.8.1.min and earlier. The vulnerabilities detected include:

* CVE-2012-6708: Selector interpreted as HTML
* CVE-2015-9251: 3rd party CORS request may execute
* CVE-2015-9251: parseHTML() executes scripts in event handlers
* CVE-2019-11358: jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution
* CVE-2020-11022: Regex in its jQuery.htmlPrefilter sometimes may introduce XSS
* CVE-2020-11023: Regex in its jQuery.htmlPrefilter sometimes may introduce XSS

The article highlights the importance of keeping third-party libraries up to date and applying security updates promptly to prevent vulnerabilities from being exploited. It also suggests reducing the attack surface by removing unused libraries. The vulnerabilities detected in jQuery are classified as:

* CWE-1104: Use of Unmaintained Third Party Components
* A9: Using Components with Known Vulnerabilities

Overall, the article emphasizes the need for proper patch management and security testing to protect web applications from third-party library vulnerabilities.

6a. Rate its clarity:

- ☐ Very Unlear
- ☐ Somewhat Unlear
- ☐ Neutral
- ☐ Somewhat Clear
- ☐ Very Clear

6b. Were you able to easily understand the security issues? ☐ Yes ☐ No

Please review the security report provided:

* The vulnerable JavaScript library is jQuery version 1.8.1.min, which has several known vulnerabilities including CVE-2012-6708 (Selector interpreted as HTML), CVE-2015-9251 (3rd party CORS request may execute), CVE-2019-11358 (jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...)), CVE-2020-11022 (Regex in its jQuery.htmlPrefilter sometimes may introduce XSS), and CVE-2020-11023 (Regex in its jQuery.htmlPrefilter sometimes may introduce XSS).
* The vulnerabilities can be exploited to hijack user accounts or introduce XSS attacks, which can lead to sensitive data breaches or other security issues.
* It is important to ensure that all available security updates are applied promptly to prevent the exploitation of known vulnerabilities in third-party libraries.
* To address this issue, a patch-management strategy should be developed to ensure that security updates are promptly applied to all third-party libraries in your application. Additionally, considering reducing the attack surface by removing any libraries that are no longer in use can also help mitigate the risk of these vulnerabilities.

7a. Rate its clarity:

- ☐ Very Unlear
- ☐ Somewhat Unlear
- ☐ Neutral
- ☐ Somewhat Clear

☐ Very Clear

7b. Were you able to easily understand the security issues? ☐ Yes ☐ No

Please review the security report provided:

The summary describes the discovery of a vulnerable JavaScript library, specifically jQuery version 1.8.1.min, which has multiple known vulnerabilities such as CVE-2012-6708, CVE-2015-9251, CVE-2015-9251, CVE-2019-11358, and CVE-2020-11022. These vulnerabilities can be used to inject malicious scripts or XSS attacks, posing a security risk to the website. It is recommended to develop a patch-management strategy to ensure prompt application of security updates to all third-party libraries in use and consider removing any unused libraries to reduce attack surface.

8a. Rate its clarity:

☐ Very Unlear

☐ Somewhat Unlear

☐ Neutral

☐ Somewhat Clear

☐ Very Clear

8b. Were you able to easily understand the security issues? ☐ Yes ☐ No

Please review the security report provided:

Summary:

The security scan identified a low-severity vulnerability in a JavaScript dependency, specifically the jQuery library version 1.8.1.min, on the website https://www.hackthissite.org. The detected vulnerabilities include issues such as selector interpretation as HTML, execution of 3rd party Cross-Origin Resource Sharing (CORS) requests, mishandling of jQuery.extend function leading to Object.prototype pollution, and potential XSS vulnerabilities introduced by regular expressions in jQuery.htmlPrefilter.

9a. Rate its clarity:

☐ Very Unlear

☐ Somewhat Unlear

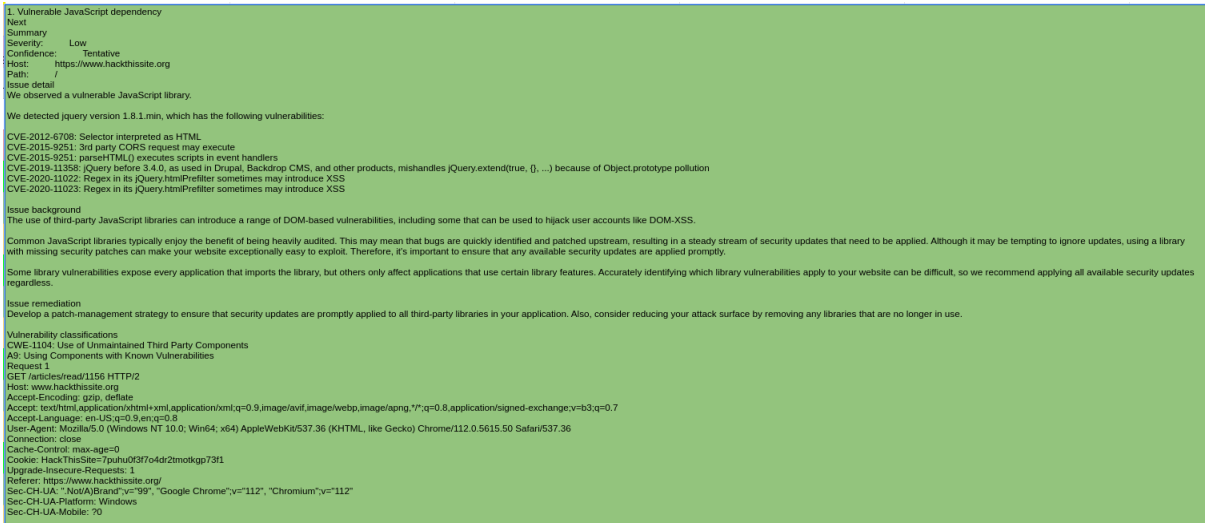
☐ Neutral

☐ Somewhat Clear

☐ Very Clear

9b. Were you able to easily understand the security issues? ☐ Yes ☐ No

Please review the security report provided:



1. Vulnerable JavaScript dependency
Next
Summary
Severity: Low
Confidence: Tentative
Host: https://www.hackthissite.org
Path: /
Issue detail
We observed a vulnerable JavaScript library.

We detected jquery version 1.8.1.min, which has the following vulnerabilities:

CVE-2012-6708: Selector interpreted as HTML
CVE-2015-9251: 3rd party CORS request may execute
CVE-2015-9251: parseHTML() executes scripts in event handlers
CVE-2019-11358: jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object.prototype pollution
CVE-2020-11022: Regex in its jQuery.htmlPrefilter sometimes may introduce XSS
CVE-2020-11023: Regex in its jQuery.htmlPrefilter sometimes may introduce XSS

Issue background
The use of third-party JavaScript libraries can introduce a range of DOM-based vulnerabilities, including some that can be used to hijack user accounts like DOM-XSS.

Common JavaScript libraries typically enjoy the benefit of being heavily audited. This may mean that bugs are quickly identified and patched upstream, resulting in a steady stream of security updates that need to be applied. Although it may be tempting to ignore updates, using a library with missing security patches can make your website exceptionally easy to exploit. Therefore, it's important to ensure that any available security updates are applied promptly.

Some library vulnerabilities expose every application that imports the library, but others only affect applications that use certain library features. Accurately identifying which library vulnerabilities apply to your website can be difficult, so we recommend applying all available security updates regardless.

Issue remediation
Develop a patch-management strategy to ensure that security updates are promptly applied to all third-party libraries in your application. Also, consider reducing your attack surface by removing any libraries that are no longer in use.

Vulnerability classifications
CWE-1104: Use of Unmaintained Third Party Components
A9: Using Components with Known Vulnerabilities
Request 1
GET /articles/read/1156 HTTP/2
Host: www.hackthissite.org
Accept-Encoding: gzip, deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Language: en-US,q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.50 Safari/537.36
Connection: close
Cache-Control: max-age=0
Cookie: HackThisSite=7puhu0f3f7o4dr2tmotkgp73f1
Upgrade-Insecure-Requests: 1
Referer: https://www.hackthissite.org/
Sec-CH-UA: ".Not/A)Brand";v="99", "Google Chrome";v="112", "Chromium";v="112"
Sec-CH-UA-Platform: Windows
Sec-CH-UA-Mobile: ?0

10a. Rate its clarity:

☐ Very Unlear

☐ Somewhat Unlear

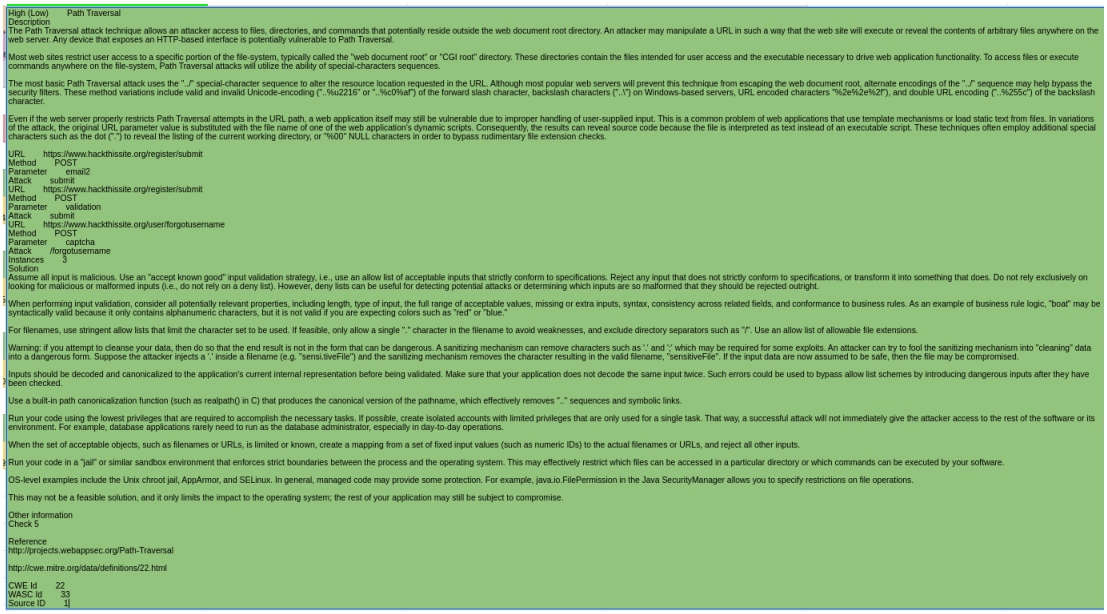
☐ Neutral

☐ Somewhat Clear

☐ Very Clear

10b. Were you able to easily understand the security issues? ☐ Yes ☐ No

Please review the security report provided:



High (Low) Path Traversal
Description
The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal.

Most web sites restrict user access to a specific portion of the file-system, typically called the "web document root" or "CGI root" directory. These directories contain the files intended for user access and the executable necessary to drive web application functionality. To access files or execute commands anywhere on the file-system, Path Traversal attacks will utilize the ability of special-characters sequences.

The most basic Path Traversal attack uses the "../" special-character sequence to alter the resource location requested in the URL. Although most popular web servers will prevent this technique from escaping the web document root, alternate encodings of the "../" sequence may help bypass the security filters. These method variations include valid and invalid Unicode-encoding ("..%u2216" or "..%c0%af") of the forward slash character, backslash characters ("..\") on Windows-based servers, URL encoded characters "%2e%2e%2f"), and double URL encoding ("..%255c") of the backslash character.

Even if the web server properly restricts Path Traversal attempts in the URL path, a web application itself may still be vulnerable due to improper handling of user-supplied input. This is a common problem of web applications that use template mechanisms or load static text from files. In variations of the attack, the original URL parameter value is substituted with the file name of one of the web application's dynamic scripts. Consequently, the results can reveal source code because the file is interpreted as text instead of an executable script. These techniques often employ additional special characters such as the dot (".") to reveal the listing of the current working directory, or "%00" NULL characters in order to bypass rudimentary file extension checks.

URL https://www.hackthissite.org/register/submit
Method POST
Parameter email2
Attack submit
URL https://www.hackthissite.org/register/submit
Method POST
Parameter validation
Attack submit
URL https://www.hackthissite.org/user/forgotusername
Method POST
Parameter captcha
Attack /forgotusername
Instances 3
Solution
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."

For filenames, use stringent allow lists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses, and exclude directory separators such as "/". Use an allow list of allowable file extensions.

Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass allow list schemes by introducing dangerous inputs after they have been checked.

Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes ".." sequences and symbolic links.

Run your code using the lowest privileges that are required to accomplish the necessary tasks. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.

Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by your software.

OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows you to specify restrictions on file operations.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.

Other information
Check 5

Reference
http://projects.webappsec.org/Path-Traversal

http://cwe.mitre.org/data/definitions/22.html

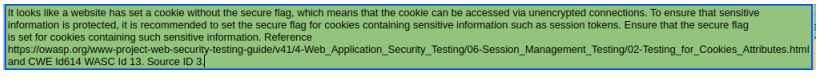
CWE Id 22
WASC Id 33
Source ID 1]

11a. Rate its clarity:

- ☐ Very Unlear
- ☐ Somewhat Unlear
- ☐ Neutral
- ☐ Somewhat Clear
- ☐ Very Clear

11b. Were you able to easily understand the security issues? ☐ Yes ☐ No

Please review the security report provided:



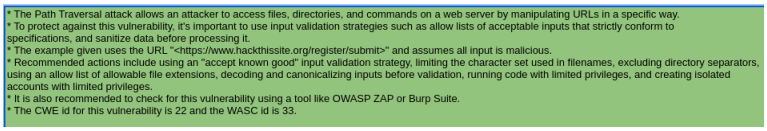
It looks like a website has set a cookie without the secure flag, which means that the cookie can be accessed via unencrypted connections. To ensure that sensitive information is protected, it is recommended to set the secure flag for cookies containing sensitive information such as session tokens. Ensure that the secure flag is set for cookies containing such sensitive information. Reference https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/06-Session_Management_Testing/02-Testing_for_Cookies_Attributes.html and CWE Id614 WASC Id 13. Source ID 3.

12a. Rate its clarity:

- ☐ Very Unlear
- ☐ Somewhat Unlear
- ☐ Neutral
- ☐ Somewhat Clear
- ☐ Very Clear

12b. Were you able to easily understand the security issues? ☐ Yes ☐ No

Please review the security report provided:


* The Path Traversal attack allows an attacker to access files, directories, and commands on a web server by manipulating URLs in a specific way.
* To protect against this vulnerability, it's important to use input validation strategies such as allow lists of acceptable inputs that strictly conform to specifications, and sanitize data before processing it.
* The example given uses the URL "<https://www.hackthissite.org/register/submit>" and assumes all input is malicious.
* Recommended actions include using an "accept known good" input validation strategy, limiting the character set used in filenames, excluding directory separators, using an allow list of allowable file extensions, decoding and canonicalizing inputs before validation, running code with limited privileges, and creating isolated accounts with limited privileges.
* It is also recommended to check for this vulnerability using a tool like OWASP ZAP or Burp Suite.
* The CWE id for this vulnerability is 22 and the WASC id is 33.

13a. Rate its clarity:

- ☐ Very Unlear
- ☐ Somewhat Unlear
- ☐ Neutral
- ☐ Somewhat Clear
- ☐ Very Clear

13b. Were you able to easily understand the security issues? ☐ Yes ☐ No

Please review the security report provided:

Path Traversal is a type of security vulnerability that allows an attacker to access files, directories, and commands outside the web document root directory. This can occur when an attacker manipulates a URL in such a way that the web server will execute or reveal the contents of arbitrary files anywhere on the web server.
To protect against Path Traversal attacks, it is important to use an "accept known good" input validation strategy, where only inputs that strictly conform to specifications are allowed. This can involve rejecting any input that does not strictly conform to specifications or transforming it into something that does. It is also important to consider all potentially relevant properties of inputs, including length, type of input, full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules.
One way to protect against Path Traversal is to use a built-in path canonicalization function, such as realpath(), which produces the canonical version of the pathname, effectively removing ".." sequences and symbolic links. Another approach is to run the code with the lowest privileges that are required to accomplish the necessary tasks, and create isolated accounts with limited privileges that are only used for a single task.
Additionally, it is recommended to use an allow list of allowable file extensions and to decoded and canonicalized inputs before validation. It is also important to use a jail or sandbox environment that enforces strict boundaries between the process and the operating system, such as chroot jail, AppArmor, or SELinux.
It is also recommended to check for the presence of the CWE-22 vulnerability in your application using the references provided, and to consult the WASC guidelines for more information on how to protect against Path Traversal attacks.

14a. Rate its clarity:

- ☐ Very Unlear
- ☐ Somewhat Unlear
- ☐ Neutral
- ☐ Somewhat Clear
- ☐ Very Clear

14b. Were you able to easily understand the security issues? ☐ Yes ☐ No

Please review the security report provided:

The vulnerability described in the summary is a Path Traversal attack, which allows an attacker to access files or directories outside of the web document root directory. This can potentially reveal sensitive information or execute arbitrary commands on the server.

To fix this vulnerability, it is recommended to use an "accept known good" input validation strategy that limits the range of acceptable inputs and rejects any input that does not strictly conform to specifications. Additionally, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

It is important to consider all potentially relevant properties when performing input validation, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. For example, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."

When dealing with filenames, using an allow list of allowable file extensions can help prevent vulnerabilities. Additionally, using a built-in path canonicalization function like realpath() in C can effectively remove ".." sequences and symbolic links from the pathname.

Running your code in a sandbox environment or creating isolated accounts with limited privileges may also provide some protection against Path Traversal attacks. It is important to ensure that your application does not decode the same input twice, as this could be used to bypass allow list schemes and compromise your software or its environment.

In general, it is recommended to limit your software's access to sensitive data and operating system resources, and use a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system.

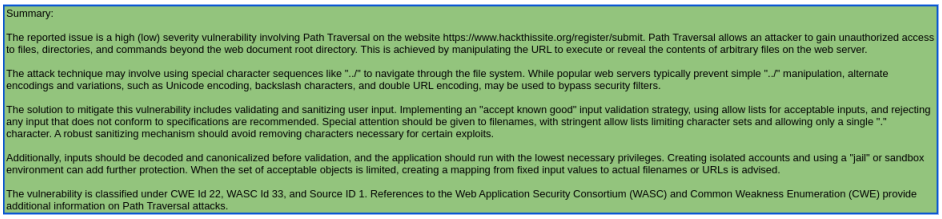
15a. Rate its clarity:

- ☐ Very Unclear
- ☐ Somewhat Unclear
- ☐ Neutral
- ☐ Somewhat Clear

☐ Very Clear

15b. Were you able to easily understand the security issues? ☐ Yes ☐ No

Please review the security report provided:



Summary:

The reported issue is a high (low) severity vulnerability involving Path Traversal on the website https://www.hackthissite.org/register/submit. Path Traversal allows an attacker to gain unauthorized access to files, directories, and commands beyond the web document root directory. This is achieved by manipulating the URL to execute or reveal the contents of arbitrary files on the web server.

The attack technique may involve using special character sequences like "../" to navigate through the file system. While popular web servers typically prevent simple "../" manipulation, alternate encodings and variations, such as Unicode encoding, backslash characters, and double URL encoding, may be used to bypass security filters.

The solution to mitigate this vulnerability includes validating and sanitizing user input. Implementing an "accept known good" input validation strategy, using allow lists for acceptable inputs, and rejecting any input that does not conform to specifications are recommended. Special attention should be given to filenames, with stringent allow lists limiting character sets and allowing only a single "." character. A robust sanitizing mechanism should avoid removing characters necessary for certain exploits.

Additionally, inputs should be decoded and canonicalized before validation, and the application should run with the lowest necessary privileges. Creating isolated accounts and using a "jail" or sandbox environment can add further protection. When the set of acceptable objects is limited, creating a mapping from fixed input values to actual filenames or URLs is advised.

The vulnerability is classified under CWE Id 22, WASC Id 33, and Source ID 1. References to the Web Application Security Consortium (WASC) and Common Weakness Enumeration (CWE) provide additional information on Path Traversal attacks.

16a. Rate its clarity:

☐ Very Unlear

☐ Somewhat Unlear

☐ Neutral

☐ Somewhat Clear

☐ Very Clear

16b. Were you able to easily understand the security issues? ☐ Yes ☐ No

17. What improvements would you suggest to make security reports more effective for software practitioners? (*Optional*)

_____

_____

18. If given a choice, which format would you prefer for receiving information about security issues in software projects? for eg: traditional reports, summarized versions of these reports, no preference, etc.

_____

_____

Appendix C

SafeAIMerge Tool Feedback

C.1 Online Survey Questionnaire

This survey aims to obtain feedback on the ZAP CI/CD integration. We are interested in understanding your perceptions of the tool, suggestions for improvement, and its potential impact on web application development. Your participation will help us design future mechanisms for integrating security testing in development workflows.

You must be 18 years or older to participate, and your participation is voluntary. You may withdraw from the survey at any time. Your responses will be kept confidential and data from this survey will be reported in aggregate. Please avoid using any names or identifying information for others in your responses. If you have any questions about the survey, you may contact Arpit Thool (arpitthool@vt.edu) or Dr. Chris Brown (dcbrown@vt.edu). Thank you for your time. We appreciate your help and support!

GitHub Repository Link for *SafeAIMerge*

Before taking the survey please watch the tool demo using the below link:

YouTube Demo Video Link

1a. Name: _____

1b. Email: _____

1c. Current Company/Organization: _____

1d. Current Role: _____

1e. Total years of technical work experience: _____

2. On a scale of 1–5, how would you rate the overall user-friendliness of the tool's interface and workflow? (1 being very less user friendly and 5 being highly user friendly)

□ 1

□ 2

□ 3

□ 4

□ 5

3. Did the LLM-generated summaries make the security findings easier to understand?

□ Yes □ No

4. What aspect of the tool's UI or output did you find most confusing or in need of improvement? (*Optional*)

_____

_____

5. How easily do you think this tool could integrate into your team's existing CI/CD pipeline?)

□ Not compatible at all

□ Significant effort required

□ Neutral

□ Some adjustments needed

□ Fits in seamlessly

6. The GitHub PR comment integration for surfacing security issues is useful for my development workflow.)

□ Strongly disagree

□ Disagree

□ Neutral

□ Agree

□ Strongly agree

7. What potential obstacles or challenges might prevent you from integrating this tool into your projects? (*Optional*)

_____

_____

8. How likely are you to adopt this tool in your own development projects if it were available?

□ Very unlikely

□ Unlikely

□ Neutral

□ Likely

□ Very likely

9. Would you recommend this CI/CD security tool to a colleague or team after today's session?" – (Yes / Maybe / No). If Maybe or No: "Please share why not, so we can address those issues. (*Optional*)

---

10. What feature or benefit of the tool do you find most compelling, and why? (*Optional*)

---

11. What improvements or additional features would increase your likelihood of using this tool? (*Optional*)

---

12. Overall, how satisfied were you with the tool demonstrated today? (Rating 1–5, 5 = extremely satisfied))

□ 1

□ 2

□ 3

□ 4

□ 5

13. Any additional comments or feedback? (*Optional*)

C.2 User Study Survey Questionnaire

This survey aims to obtain feedback on the ZAP CI/CD integration. We are interested in understanding your perceptions of the tool, suggestions for improvement, and its potential impact on web application development. Your participation will help us design future mechanisms for integrating security testing in development workflows.

You must be 18 years or older to participate, and your participation is voluntary. You may withdraw from the survey at any time. Your responses will be kept confidential and data from this survey will be reported in aggregate. Please avoid using any names or identifying information for others in your responses. If you have any questions about the survey, you may contact Arpit Thool (arpitthool@vt.edu) or Dr. Chris Brown (dcbrown@vt.edu). Thank you for your time. We appreciate your help and support!

What you'll do (45–60 minutes):

1. Integrate ZAP scans into a GitHub repo and fix as many vulnerabilities as possible (15-20 min) by seeing the security report.

2. Use our new CI/CD-integrated security tool on a second repo and try to fix as many vulnerabilities as possible (15-20 min) by seeing the security report.

GitHub Repository Link for *SafeAIMerge*

1a. Name: _____

1b. Email: _____

1c. Current Company/Organization: _____

1d. Current Role: _____

1e. Total years of technical work experience: _____

1e. Experience with CI/CD pipelines?

☐ Yes ☐ No

C.2.1 NASA-TLX (Mental Workload Index) [Baseline ZAP]

2a. How mentally demanding was the task?

☐ Very Low

☐ Low

☐ Neutral

☐ High

☐ Very High

2b. How physically demanding was the task?

☐ Very Low

☐ Low

☐ Neutral

☐ High

☐ Very High

2c. How hurried or rushed was the task?

☐ Very Low

☐ Low

☐ Neutral

☐ High

☐ Very High

2d. How successful were you in accomplishing what you were asked to do?

☐ Highly Unsuccessful

☐ Unsuccessful

☐ Neutral

☐ Successful

☐ Highly Successful

2e. How hard did you have to work to accomplish your level of performance?

☐ Very Low

☐ Low

☐ Neutral

☐ High

☐ Very High

2f. How insecure, discouraged, irritated, stressed, and annoyed were you?

☐ Very Less Frustrated

☐ Less Frustrated

☐ Neutral

☐ Highly Frustrated

☐ Very Highly Frustrated

C.2.2 NASA-TLX (Mental Workload Index) [SafeAIMerge]

3a. How mentally demanding was the task?

☐ Very Low

☐ Low

☐ Neutral

☐ High

☐ Very High

3b. How physically demanding was the task?

☐ Very Low

☐ Low

☐ Neutral

☐ High

☐ Very High

3c. How hurried or rushed was the task?

☐ Very Low

☐ Low

☐ Neutral

☐ High

☐ Very High

3d. How successful were you in accomplishing what you were asked to do?

☐ Highly Unsuccessful

☐ Unsuccessful

☐ Neutral

☐ Successful

☐ Highly Successful

3e. How hard did you have to work to accomplish your level of performance?

☐ Very Low

- ☐ Low
- ☐ Neutral
- ☐ High
- ☐ Very High

3f. How insecure, discouraged, irritated, stressed, and annoyed were you?

- ☐ Very Less Frustrated
- ☐ Less Frustrated
- ☐ Neutral
- ☐ Highly Frustrated
- ☐ Very Highly Frustrated

C.2.3 Comparative Questions (After Completing Both Sections)

4a. Overall, which workflow did you find easier to use?

- ☐ Baseline: ZAP Workflow
- ☐ Tool-Integrated (SafeAIMerge) Workflow
- ☐ Both equally
- ☐ Neither

4b. Which workflow helped you better understand the vulnerability and how to fix it?

- ☐ Baseline: ZAP Workflow
- ☐ Tool-Integrated (SafeAIMerge) Workflow
- ☐ Both equally
- ☐ Neither

4c. Which workflow would you prefer to use?

- ☐ Baseline: ZAP Workflow

- ☐ Tool-Integrated (SafeAIMerge) Workflow
- ☐ Both equally
- ☐ Neither

4d. Which workflow produced clearer or more actionable scan reports?

- ☐ Baseline: ZAP Workflow
- ☐ Tool-Integrated (SafeAIMerge) Workflow
- ☐ Both equally
- ☐ Neither

4e. Which workflow took less time to interpret vulnerabilities?

- ☐ Baseline: ZAP Workflow
- ☐ Tool-Integrated (SafeAIMerge) Workflow
- ☐ Both equally
- ☐ Neither

Bibliography

[1] J. Beckman, "Agile statistics: How many companies use agile in 2023?," *Tech Report*, 2024. https://techreport.com/statistics/business-workplace/how-many-companies-use-agile/.

[2] P. Rodríguez, M. Mäntylä, M. Oivo, L. E. Lwakatare, P. Seppänen, and P. Kuvaja, "Advances in using agile and lean processes for software development," in *Advances in Computers*, vol. 113, pp. 135–224, Elsevier, 2019.

[3] M. Awad, "A comparison between agile and traditional software development methodologies," *University of Western Australia*, vol. 30, pp. 1–69, 2005.

[4] D. Albuquerque, E. Guimaraes, M. Perkusich, A. Costa, E. Dantas, F. Ramos, and H. Almeida, "Defining agile requirements change management: a mapping study," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pp. 1421–1424, 2020.

[5] D. L. Buresh, "Customer satisfaction and agile methods," *IEEE Reliability Society Annual Technology Report*, pp. 1–8, 2008.

[6] M. Tatam, B. Shanmugam, S. Azam, and K. Kannoorpatti, "A review of threat modelling approaches for apt-style attacks," *Heliyon*, vol. 7, no. 1, 2021.

[7] K. Rindell, S. Hyrynsalmi, and V. Leppänen, "Aligning security objectives with agile software development," in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, pp. 1–9, 2018.

[8] A. Azanha, A. R. T. T. Argoud, J. B. d. Camargo, and P. D. Antoniolli, "Agile project management with scrum," *International Journal of Managing Projects in Business*, vol. 10, pp. 121–142, 2017.

[9] J. Li, Y. Zhang, X. Chen, and Y. Xiang, "Secure attribute-based data sharing for resource-limited users in cloud computing," *computers & security*, vol. 72, pp. 1–12, 2018.

[10] A. Chowdhury, "Recent cyber security attacks and their mitigation approaches–an overview," in *Applications and Techniques in Information Security: 6th International Conference, ATIS 2016, Cairns, QLD, Australia, October 26-28, 2016, Proceedings 7*, pp. 54–65, Springer, 2016.

[11] N. Boltz, L. Sterz, C. Gerking, and O. Raabe, "A model-based framework for simplified collaboration of legal and software experts in data protection assessments," 2022.

[12] "Security benefits for agile software development," *Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering*, 2017.

[13] G. Boström, J. Wäyrynen, M. Bodén, K. Beznosov, and P. Kruchten, "Extending xp practices to support security requirements engineering," *Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems*, pp. 11–18, 2006.

[14] A. F. Arbain, I. Ghani, and S. R. Jeong, "A systematic literature review on secure software development using feature driven development (fdd) agile model," *Journal of Korean Society for Internet Information*, vol. 15, pp. 13–27, 2014.

[15] H. Keramati and S.-H. Mirian-Hosseinabadi, "Integrating software development security activities with agile methodologies," in *2008 IEEE/ACS International Conference on Computer Systems and Applications*, pp. 749–754, IEEE, 2008.

[16] R. Hu, Z. Wang, J. Hu, J. Xu, and J. Xie, "Agile web development with web framework," in *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1–4, IEEE, 2008.

[17] R. E. Bucklin and C. Sismeiro, "Click here for internet insight: Advances in clickstream data analysis in marketing," *Journal of Interactive marketing*, vol. 23, no. 1, pp. 35–48, 2009.

[18] L. Dencheva, *Comparative analysis of Static application security testing (SAST) and Dynamic application security testing (DAST) by using open-source web application penetration testing tools*. PhD thesis, Dublin, National College of Ireland, 2022.

[19] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices," *IEEE access*, vol. 5, pp. 3909–3943, 2017.

[20] L. Braz and A. Bacchelli, "Software security during modern code review: The developer's perspective," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 810–821, 2022.

[21] T. Memmel, F. Gundelsweiler, and H. Reiterer, "Agile human-centered software engineering," 2007.

[22] C. Weir, A. Rashid, and J. Noble, "Challenging software developers: dialectic as a foundation for security assurance techniques," *Journal of Cybersecurity*, vol. 6, no. 1, p. tyaa007, 2020.

[23] J. d. V. Mohino, J. B. Higuera, J. R. B. Higuera, and J. A. S. Montalvo, "The applica-

tion of a new secure software development life cycle (s-sdlc) with agile methodologies," *Electronics*, vol. 8, p. 1218, 2019.

[24] Q. M. Yas, A. Alazzawi, and B. Rahmatullah, "A comprehensive review of software development life cycle methodologies: Pros, cons, and future directions," *Iraqi Journal for Computer Science and Mathematics*, pp. 173–190, 2023.

[25] S. Pargaonkar, "Advancements in security testing: A comprehensive review of methodologies and emerging trends in software quality engineering," *International Journal of Science and Research (IJSR)*, vol. 12, no. 9, pp. 61–66, 2023.

[26] K. R. Riisom, M. S. Hubel, H. M. Alradhi, N. B. Nielsen, K. Kuusinen, and R. Jabangwe, "Software security in agile software development: A literature review of challenges and solutions," in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, pp. 1–5, 2018.

[27] S. M. Furnell, A. Jusoh, and D. Katsabas, "The challenges of understanding and using security: A survey of end-users," *Computers & Security*, vol. 25, no. 1, pp. 27–35, 2006.

[28] J. Smith, L. N. Q. Do, and E. Murphy-Hill, "Why can't johnny fix vulnerabilities: A usability evaluation of static analysis tools for security," in *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pp. 221–238, 2020.

[29] A. O. Airhiavbere and D. N. Idehen, "Secure software engineering: A synthesis of ssdlc, devsecops, and ai-driven threat mitigation," *International Journal of Development Mathematics (IJDM)*, vol. 2, pp. 188–199, 2025.

[30] J. Wäyrynen, M. Bodén, and G. Boström, "Security engineering and extreme programming: An impossible marriage?," in *Conference on Extreme Programming and Agile Methods*, pp. 117–128, Springer, 2004.

[31] A. Tubis, S. Werbińska-Wojciechowska, M. Góralczyk, A. Wróblewski, and B. Ziętek, "Cyber-attacks risk analysis method for different levels of automation of mining processes in mines based on fuzzy theory use," *Sensors*, vol. 20, p. 7210, 2020.

[32] B. Boehm and P. Behnamghader, "Anticipatory development processes for reducing total ownership costs and schedules," *Systems Engineering*, vol. 22, pp. 401–410, 2019.

[33] R. Hoda, N. Salleh, and J. Grundy, "The rise and evolution of agile software development," *IEEE software*, vol. 35, no. 5, pp. 58–63, 2018.

[34] O. O. Abiona, O. J. Oladapo, O. T. Modupe, O. C. Oyeniran, A. O. Adewusi, and A. M. Komolafe, "The emergence and importance of devsecops: Integrating and reviewing security practices within the devops pipeline," *World Journal of Advanced Engineering Technology and Sciences*, vol. 11, no. 2, pp. 127–133, 2024.

[35] K. Rindell, S. Hyrynsalmi, and V. Leppänen, "Busting a myth: Review of agile security engineering methods," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, pp. 1–10, 2017.

[36] R. Jabangwe, K. Kuusinen, K. R. Riisom, M. S. Hubel, H. M. Alradhi, and N. B. Nielsen, "Challenges and solutions for addressing software security in agile software development: A literature review and rigor and relevance assessment," *Research Anthology on Recent Trends, Tools, and Implications of Computer Programming*, pp. 1875–1888, 2021.

[37] K. Beznosov and P. Kruchten, "Towards agile security assurance," in *Proceedings of the 2004 workshop on New security paradigms*, pp. 47–54, 2004.

[38] S. Bartsch, "Practitioners' perspectives on security in agile development," in *2011 Sixth*

International Conference on Availability, Reliability and Security, pp. 479–484, IEEE, 2011.

[39] M. Hammad, I. Inayat, and M. Zahid, "Risk management in agile software development: A survey," in *2019 international conference on frontiers of information technology (fit)*, pp. 162–1624, IEEE, 2019.

[40] A. Agrawal, M. A. Atiq, and L. Maurya, "A current study on the limitations of agile methods in industry using secure google forms," *Procedia Computer Science*, vol. 78, pp. 291–297, 2016.

[41] H. Assal and S. Chiasson, "Security in the software development lifecycle," in *Fourteenth symposium on usable privacy and security (SOUPS 2018)*, pp. 281–296, 2018.

[42] T. Ayalew, T. Kidane, and B. Carlsson, "Identification and evaluation of security activities in agile projects," in *Secure IT Systems: 18th Nordic Conference, NordSec 2013, Ilulissat, Greenland, October 18-21, 2013, Proceedings 18*, pp. 139–153, Springer, 2013.

[43] X. Ge, R. F. Paige, F. A. Polack, H. Chivers, and P. J. Brooke, "Agile development of secure web applications," in *Proceedings of the 6th international conference on Web engineering*, pp. 305–312, 2006.

[44] L. Williams, "Secure software lifecycle," *The Cyber Security Body of Knowledge*, 2019.

[45] 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," in *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, pp. 145–154, IEEE, 2020.

[46] A. Koskinen, "Devsecops: building security into the core of devops," 2019.

[47] P. Maier, Z. Ma, and R. Bloem, "Towards a secure scrum process for agile web application development," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, pp. 1–8, 2017.

[48] M. Zaydi, Y. Maleh, H. Zaydi, Y. Khourdifi, B. Nassereddine, and Z. Bakouri, "Agile security and compliance integration," *Agile Security in the Digital Era: Challenges and Cybersecurity Trends*, p. 68, 2024.

[49] P. Bajpai and A. Lewis, "Secure development workflows in ci/cd pipelines," in *2022 IEEE Secure Development Conference (SecDev)*, pp. 65–66, 2022.

[50] E. Lebanidze, "Securing enterprise web applications at the source: An application security perspective perspective," 2006.

[51] A. A. Ur Rahman and L. Williams, "Software security in devops: synthesizing practitioners' perceptions and practices," in *Proceedings of the international workshop on continuous software evolution and delivery*, pp. 70–76, 2016.

[52] D. Ashenden and G. Ollis, "Putting the sec in devsecops: Using social practice theory to improve secure software development," in *New Security Paradigms Workshop 2020*, pp. 34–44, 2020.

[53] F. Spotswood, T. Chatterton, A. Tapp, and D. Williams, "Analysing cycling as a social practice: An empirical grounding for behaviour change," *Transportation research part F: traffic psychology and behaviour*, vol. 29, pp. 22–33, 2015.

[54] H. Taherdoost, "A review of technology acceptance and adoption models and theories," *Procedia manufacturing*, vol. 22, pp. 960–967, 2018.

[55] P. Mohagheghi, W. Gilani, A. Stefanescu, and M. A. Fernandez, "An empirical study of

the state of the practice and acceptance of model-driven engineering in four industrial cases," *Empirical software engineering*, vol. 18, no. 1, pp. 89–116, 2013.

[56] S. Kent, "Model driven engineering," in *International conference on integrated formal methods*, pp. 286–298, Springer, 2002.

[57] A. Senarath, M. Grobler, and N. A. G. Arachchilage, "Will they use it or not? investigating software developers' intention to follow privacy engineering methodologies," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 4, pp. 1–30, 2019.

[58] S. Gürses and J. M. Del Alamo, "Privacy engineering: Shaping an emerging field of research and practice," *IEEE Security & Privacy*, vol. 14, no. 2, pp. 40–46, 2016.

[59] C. K. Riemenschneider, B. C. Hardgrave, and F. D. Davis, "Explaining software developer acceptance of methodologies: a comparison of five theoretical models," *IEEE transactions on Software Engineering*, vol. 28, no. 12, pp. 1135–1145, 2002.

[60] W. Kasri, Y. Himeur, H. A. Alkhazaleh, S. Tarapiah, S. Atalla, W. Mansoor, and H. Al-Ahmad, "From vulnerability to defense: The role of large language models in enhancing cybersecurity," *Computation*, vol. 13, p. 30, 2025.

[61] S. M. Taghavi and F. Feyzi, "Using large language models to better detect and handle software vulnerabilities and cyber security threats," 2024.

[62] H. Szmurlo and Z. Akhtar, "Digital sentinels and antagonists: The dual nature of chatbots in cybersecurity," *Information*, vol. 15, p. 443, 2024.

[63] M. Charfeddine, H. M. Kammoun, B. Hamdaoui, and M. Guizani, "Chatgpt's security risks and benefits: Offensive and defensive use-cases, mitigation measures, and future implications," *IEEE Access*, vol. 12, pp. 30263–30310, 2024.

[64] S. Dupont, G. Ginis, M. Malacario, C. Porretti, N. Maunero, C. Ponsard, and P. Massonet, "Incremental common criteria certification processes using devsecops practices," in *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 12–23, IEEE, 2021.

[65] R. Andrade, J. Torres, and I. Ortiz-Garcés, "Enhancing security in software design patterns and antipatterns: A framework for llm-based detection," *Electronics*, vol. 14, no. 3, p. 586, 2025.

[66] M. Keltek, R. Hu, M. F. Sani, and Z. Li, "Boosting cybersecurity vulnerability scanning based on llm-supported static application security testing," *arXiv preprint arXiv:2409.15735*, 2024.

[67] A. Lekssays, H. Mouhcine, K. Tran, T. Yu, and I. Khalil, "{LLMxCPG}:{Context-Aware} vulnerability detection through code property {Graph-Guided} large language models," in *34th USENIX Security Symposium (USENIX Security 25)*, pp. 489–507, 2025.

[68] K. Shashwat, F. Hahn, X. Ou, D. Goldgof, L. Hall, J. Ligatti, S. R. Rajgopalan, and A. Z. Tabari, "A preliminary study on using large language models in software pentesting," *arXiv preprint arXiv:2401.17459*, 2024.

[69] J. Wagner, "Analysis of security findings and reduction of false positives through large language models," 2024.

[70] X. Zhou, D.-M. Tran, T. Le-Cong, T. Zhang, I. C. Irsan, J. Sumarlin, B. Le, and D. Lo, "Comparison of static application security testing tools and large language models for repo-level vulnerability detection," *arXiv preprint arXiv:2407.16235*, 2024.

[71] B. Steenhoek, K. Sivaraman, R. S. Gonzalez, Y. Mohylevskyy, R. Z. Moghaddam,

and W. Le, "Closing the gap: A user study on the real-world usefulness of ai-powered vulnerability detection & repair in the ide," *arXiv preprint arXiv:2412.14306*, 2024.

[72] J. Wang, Y. Shao, Y. Ge, and R. Yu, "A survey of vehicle to everything (v2x) testing," *Sensors*, vol. 19, no. 2, p. 334, 2019.

[73] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "{PentestGPT}: Evaluating and harnessing large language models for automated penetration testing," in *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 847–864, 2024.

[74] H. Wang and B. Hooi, "Automated phishing detection using urls and webpages," *arXiv preprint arXiv:2408.01667*, 2024.

[75] F. Trad and A. Chehab, "Prompt engineering or fine-tuning? a case study on phishing detection with large language models," *Machine Learning and Knowledge Extraction*, vol. 6, no. 1, pp. 367–384, 2024.

[76] S. Mahendru and T. Pandit, "Securenet: A comparative study of deberta and large language models for phishing detection," in *2024 IEEE 7th International Conference on Big Data and Artificial Intelligence (BDAI)*, pp. 160–169, IEEE, 2024.

[77] P. M. S. Sánchez, A. H. Celdrán, G. Bovet, and G. M. Pérez, "Transfer learning in pre-trained large language models for malware detection based on system calls," in *MILCOM 2024-2024 IEEE Military Communications Conference (MILCOM)*, pp. 853–858, IEEE, 2024.

[78] J. L. Hu, M. Ebrahimi, and H. Chen, "Single-shot black-box adversarial attacks against malware detectors: A causal language model approach," in *2021 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 1–6, IEEE, 2021.

[79] N. Zahan, P. Burckhardt, M. Lysenko, F. Aboukhadijeh, and L. Williams, "Leveraging large language models to detect npm malicious packages," *arXiv preprint arXiv:2403.12196*, 2024.

[80] J. Sun, J. Chen, Z. Xing, Q. Lu, X. Xu, and L. Zhu, "Where is it? tracing the vulnerability-relevant files from vulnerability reports," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13, 2024.

[81] S. Sakaoglu, "Kartal: Web application vulnerability hunting using large language models: Novel method for detecting logical vulnerabilities in web applications with fine-tuned large language models," 2023.

[82] A. Shestov, R. Levichev, R. Mussabayev, E. Maslov, P. Zadorozhny, A. Cheshkov, R. Mussabayev, A. Toleu, G. Tolegen, and A. Krassovitskiy, "Finetuning large language models for vulnerability detection," *IEEE Access*, 2025.

[83] J. Wang, Z. Huang, H. Liu, N. Yang, and Y. Xiao, "Defecthunter: A novel llm-driven boosted-conformer-based code vulnerability detection mechanism," *arXiv preprint arXiv:2309.15324*, 2023.

[84] J. Gonçalves, T. Dias, E. Maia, and I. Praça, "Scope: Evaluating llms for software vulnerability detection," in *International Symposium on Distributed Computing and Artificial Intelligence*, pp. 34–43, Springer, 2024.

[85] X. Du, G. Zheng, K. Wang, Y. Zou, Y. Wang, W. Deng, J. Feng, M. Liu, B. Chen, X. Peng, *et al.*, "Vul-rag: Enhancing llm-based vulnerability detection via knowledge-level rag," *arXiv preprint arXiv:2406.11147*, 2024.

[86] N. S. Mathews, Y. Brus, Y. Aafer, M. Nagappan, and S. McIntosh, "Llbezpeky:

Leveraging large language models for vulnerability detection," *arXiv preprint arXiv:2401.01269*, 2024.

[87] C. Zhang, H. Liu, J. Zeng, K. Yang, Y. Li, and H. Li, "Prompt-enhanced software vulnerability detection using chatgpt," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, pp. 276–277, 2024.

[88] P. Ince, X. Luo, J. Yu, J. K. Liu, and X. Du, "Detect llama-finding vulnerabilities in smart contracts using large language models," in *Australasian Conference on Information Security and Privacy*, pp. 424–443, Springer, 2024.

[89] B. Bokkena, "Enhancing it security with llm-powered predictive threat intelligence," in *2024 5th International Conference on Smart Electronics and Communication (ICOSEC)*, pp. 751–756, IEEE, 2024.

[90] M. A. Ferrag, F. Alwahedi, A. Battah, B. Cherif, A. Mechri, and N. Tihanyi, "Generative ai and large language models for cyber security: All insights you need," *Available at SSRN 4853709*, 2024.

[91] J. Zhang, H. Bu, H. Wen, Y. Liu, H. Fei, R. Xi, L. Li, Y. Yang, H. Zhu, and D. Meng, "When llms meet cybersecurity: A systematic literature review," *Cybersecurity*, vol. 8, no. 1, p. 55, 2025.

[92] J. Wang, X. Luo, L. Cao, H. He, H. Huang, J. Xie, A. Jatowt, and Y. Cai, "Is your ai-generated code really safe? evaluating large language models on secure code generation with codeseceval," *arXiv preprint arXiv:2407.02395*, 2024.

[93] J. He, M. Vero, G. Krasnopolska, and M. Vechev, "Instruction tuning for secure code generation," *arXiv preprint arXiv:2402.09497*, 2024.

[94] J. Li, F. Rabbi, C. Cheng, A. Sangalay, Y. Tian, and J. Yang, "An exploratory study on fine-tuning large language models for secure code generation," *arXiv preprint arXiv:2408.09078*, 2024.

[95] G. Li, C. Zhi, J. Chen, J. Han, and S. Deng, "Exploring parameter-efficient fine-tuning of large language model on automated program repair," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 719–731, 2024.

[96] M. Jin, S. Shahriar, M. Tufano, X. Shi, S. Lu, N. Sundaresan, and A. Svyatkovskiy, "Inferfix: End-to-end program repair with llms," in *Proceedings of the 31st ACM joint european software engineering conference and symposium on the foundations of software engineering*, pp. 1646–1656, 2023.

[97] J. Zhao, D. Yang, L. Zhang, X. Lian, Z. Yang, and F. Liu, "Enhancing automated program repair with solution design," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1706–1718, 2024.

[98] J. Kong, X. Xie, M. Cheng, S. Liu, X. Du, and Q. Guo, "Contrastrepair: Enhancing conversation-based automated program repair via contrastive test case pairs," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 8, pp. 1–31, 2025.

[99] A. Z. Yang, C. Le Goues, R. Martins, and V. Hellendoorn, "Large language models for test-free fault localization," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pp. 1–12, 2024.

[100] M. L. Kuhn, "147 million social security numbers for sale: Developing data protection legislation after mass cybersecurity breaches," *Iowa L. Rev.*, vol. 104, p. 417, 2018.

[101] M. Hölzl, A. Rauschmayer, and M. Wirsing, "Engineering of software-intensive systems: State of the art and research challenges," *Software-Intensive Systems and New Computing Paradigms: Challenges and Visions*, pp. 1–44, 2008.

[102] Y. Pan, "Interactive application security testing," in *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, pp. 558–561, IEEE, 2019.

[103] F. Mateo Tudela, J.-R. Bermejo Higuera, J. Bermejo Higuera, J.-A. Sicilia Montalvo, and M. I. Argyros, "On combining static, dynamic and interactive analysis security testing tools to improve owasp top ten security vulnerability detection in web applications," *Applied Sciences*, vol. 10, no. 24, p. 9119, 2020.

[104] M. M. Casanova Páez, "Application security testing tools study and proposal," 2021.

[105] K. Sinchana, C. Sinchana, H. Gururaj, and B. S. Kumar, "Performance evaluation and analysis of various network security tools," in *2019 International Conference on Communication and Electronics Systems (ICCES)*, pp. 644–650, IEEE, 2019.

[106] A. Kore, T. Hinduja, A. Sawant, S. Indorkar, S. Wagh, and S. Rankhambe, "Burp suite extension for script based attacks for web applications," in *2022 6th International Conference on Electronics, Communication and Aerospace Technology*, pp. 651–657, IEEE, 2022.

[107] M. Zhang, G. Jiang, S. Liu, J. Chen, and M. Zhang, "Llm–assisted data augmentation for chinese dialogue–level dependency parsing," *Computational Linguistics*, pp. 1–24, 2024.

[108] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on large language model (llm) security and privacy: The good, the bad, and the ugly," *High-Confidence Computing*, p. 100211, 2024.

[109] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[110] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 2020.

[111] B. Meskó and E. J. Topol, "The imperative for regulatory oversight of large language models (or generative ai) in healthcare," *NPJ digital medicine*, vol. 6, no. 1, p. 120, 2023.

[112] L. Li, Y. Zhang, and L. Chen, "Prompt distillation for efficient llm-based recommendation," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 1348–1357, 2023.

[113] X. Lin, W. Wang, Y. Li, S. Yang, F. Feng, Y. Wei, and T.-S. Chua, "Data-efficient fine-tuning for llm-based recommendation," *arXiv preprint arXiv:2401.17197*, 2024.

[114] C. Zan, L. Ding, L. Shen, Y. Zhen, W. Liu, and D. Tao, "Building accurate translation-tailored llms with language aware instruction tuning," *arXiv preprint arXiv:2403.14399*, 2024.

[115] X. Pu, M. Gao, and X. Wan, "Summarization is (almost) dead," *arXiv preprint arXiv:2309.09558*, 2023.

[116] W. Zhang, Y. Deng, B. Liu, S. J. Pan, and L. Bing, "Sentiment analysis in the era of large language models: A reality check," *arXiv preprint arXiv:2305.15005*, 2023.

[117] D. Allemang and J. Sequeda, "Increasing the llm accuracy for question answering: Ontologies to the rescue!," *arXiv preprint arXiv:2405.11706*, 2024.

[118] J. Yang, H. B. Li, and D. Wei, "The impact of chatgpt and llms on medical imaging stakeholders: perspectives and use cases," *Meta-Radiology*, p. 100007, 2023.

[119] I. de Zarzà, J. de Curtò, G. Roig, and C. T. Calafate, "Optimized financial planning: Integrating individual and cooperative budgeting models with llm recommendations," *AI*, vol. 5, no. 1, pp. 91–114, 2023.

[120] C. Cui, Y. Ma, X. Cao, W. Ye, and Z. Wang, "Drive as you speak: Enabling human-like interaction with large language models in autonomous vehicles," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 902–909, 2024.

[121] A. Moniruzzaman and D. S. A. Hossain, "Comparative study on agile software development methodologies," *arXiv preprint arXiv:1307.3356*, 2013.

[122] S. Al-Saqqa, S. Sawalha, and H. AbdelNabi, "Agile software development: Methodologies and trends.," *International Journal of Interactive Mobile Technologies*, vol. 14, no. 11, 2020.

[123] K. Jammalamadaka and V. R. Krishna, "Agile software development and challenges," *International Journal of Research in Engineering and Technology*, vol. 2, no. 08, pp. 125–129, 2013.

[124] K. Conboy, "Agility from first principles: Reconstructing the concept of agility in information systems development," *Information systems research*, vol. 20, no. 3, pp. 329–354, 2009.

[125] S. Xiao, J. Witschey, and E. Murphy-Hill, "Social influences on secure development tool adoption: why security tools spread," in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pp. 1095–1106, 2014.

[126] A. Edmundson, B. Holtkamp, E. Rivera, M. Finifter, A. Mettler, and D. Wagner, "An empirical study on the effectiveness of security code review," in *Engineering Secure Software and Systems: 5th International Symposium, ESSoS 2013, Paris, France, February 27-March 1, 2013. Proceedings 5*, pp. 197–212, Springer, 2013.

[127] P. D. Chowdhury, J. Hallett, N. Patnaik, M. Tahaei, and A. Rashid, "Developers are neither enemies nor users: they are collaborators," in *2021 IEEE Secure Development Conference (SecDev)*, pp. 47–55, IEEE, 2021.

[128] P. Schneider, M. Voggenreiter, A. Gulraiz, and F. Matthes, "Semantic similarity-based clustering of findings from security testing tools," *arXiv preprint arXiv:2211.11057*, 2022.

[129] R. N. Rajapakse, M. Zahedi, and M. A. Babar, "An empirical analysis of practitioners' perspectives on security tool integration into devops," in *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 1–12, 2021.

[130] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requirements engineering*, vol. 10, pp. 34–44, 2005.

[131] J. Smith, C. Theisen, and T. Barik, "A case study of software security red teams at microsoft," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 1–10, IEEE, 2020.

[132] L. Williams, G. McGraw, and S. Migues, "Engineering security vulnerability prevention, detection, and response," *IEEE Software*, vol. 35, no. 5, pp. 76–80, 2018.

[133] M. Howard and S. Lipner, *The security development lifecycle*, vol. 8. Microsoft Press Redmond, 2006.

[134] D. Baca and B. Carlsson, "Agile development with security engineering activities," in *Proceedings of the 2011 international conference on software and systems process*, pp. 149–158, 2011.

[135] M. Swanson, J. Hash, and P. Bowen, "Guide for developing security plans for federal information systems." National Institute of Standards and Technology, 2006. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-18r1.pdf.

[136] E. Wińska and W. Dąbrowski, "Software development artifacts in large agile organizations: a comparison of scaling agile methods," *Data-Centric Business and Applications: Towards Software Development (Volume 4)*, pp. 101–116, 2020.

[137] M. L. Junior and M. Godinho Filho, "Variations of the kanban system: Literature review and classification," *International journal of production economics*, vol. 125, no. 1, pp. 13–21, 2010.

[138] O. Al-Baik and J. Miller, "The kanban approach, between agility and leanness: a systematic review," *Empirical Software Engineering*, vol. 20, pp. 1861–1897, 2015.

[139] C. Hofmann, S. Lauber, B. Haefner, and G. Lanza, "Development of an agile development method based on kanban for distributed part-time teams and an introduction framework," *Procedia Manufacturing*, vol. 23, pp. 45–50, 2018.

[140] A. Hosseini, H. A. Kishawy, and H. M. Hussein, "Lean manufacturing," *Modern Manufacturing Engineering*, pp. 249–269, 2015.

[141] M. K. Yacoub, M. A. A. Mostafa, and A. B. Farid, "A new approach for distributed software engineering teams based on kanban method for reducing dependency.," *J. Softw.*, vol. 11, no. 12, pp. 1231–1241, 2016.

[142] S. Malakar, *Agile Methodologies In-Depth: Delivering Proven Agile, SCRUM and Kanban Practices for High-Quality Business Demands (English Edition)*. BPB Publications, 2021.

[143] J. Saltz and R. Heckman, "Exploring which agile principles students internalize when using a kanban process methodology," *Journal of Information Systems Education*, vol. 31, no. 1, p. 51, 2020.

[144] N. Cvetković, S. Morača, M. Jovanović, M. Medojević, and B. Lalić, "Enhancing the agility and performances of a project with lean manufacturing practices.," *Annals of DAAAM & Proceedings*, vol. 28, 2017.

[145] S. Arachchi and I. Perera, "Continuous integration and continuous delivery pipeline automation for agile software project management," in *2018 Moratuwa Engineering Research Conference (MERCon)*, pp. 156–161, IEEE, 2018.

[146] C. Singh, N. S. Gaba, M. Kaur, and B. Kaur, "Comparison of different ci/cd tools integrated with cloud platform," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 7–12, IEEE, 2019.

[147] A. MUSTYALA, "Ci/cd pipelines in kubernetes: Accelerating software development and deployment," *EPH-International Journal of Science And Engineering*, vol. 8, no. 3, pp. 1–11, 2022.

[148] K. Petersen and C. Wohlin, "The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study," *Empirical Software Engineering*, vol. 15, pp. 654–693, 2010.

[149] G. Melnik and F. Maurer, "A cross-program investigation of students' perceptions

of agile methods," in *Proceedings of the 27th international Conference on Software Engineering*, pp. 481–488, 2005.

[150] K. M. Goertzel, T. Winograd, H. L. McKinley, L. Oh, M. Colon, T. McGibbon, E. Fedchak, and R. Vienneau, "Software security assurance: a state-of-art report (sar)," *DTIC Document*, 2007.

[151] L. E. Lwakatare, P. Kuvaja, and M. Oivo, "Relationship of devops to agile, lean and continuous deployment: A multivocal literature review study," in *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings 17*, pp. 399–415, Springer, 2016.

[152] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, "What is devops? a systematic mapping study on definitions and practices," in *Proceedings of the scientific workshop proceedings of XP2016*, pp. 1–11, 2016.

[153] R. Mao, H. Zhang, Q. Dai, H. Huang, G. Rong, H. Shen, L. Chen, and K. Lu, "Preliminary findings about devsecops from grey literature," in *2020 IEEE 20th international conference on software quality, reliability and security (QRS)*, pp. 450–457, IEEE, 2020.

[154] S. K. T. Ziauddin and S. Zia, "An effort estimation model for agile software development," *Advances in computer science and its applications (ACSA)*, vol. 2, no. 1, pp. 314–324, 2012.

[155] V. Kongsli, "Towards agile security in web applications," in *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pp. 805–808, 2006.

[156] B. Chess and G. McGraw, "Static analysis for security," *IEEE security & privacy*, vol. 2, no. 6, pp. 76–79, 2004.

[157] P. Salini and S. Kanmani, "Survey and analysis on security requirements engineering," *Computers & Electrical Engineering*, vol. 38, no. 6, pp. 1785–1797, 2012.

[158] L. ben Othmane, P. Angin, H. Weffers, and B. Bhargava, "Extending the agile development process to develop acceptably secure software," *IEEE Transactions on dependable and secure computing*, vol. 11, no. 6, pp. 497–509, 2014.

[159] A. Firdaus, I. Ghani, and S. R. Jeong, "Secure feature driven development (sfdd) model for secure software development," *Procedia-Social and Behavioral Sciences*, vol. 129, pp. 546–553, 2014.

[160] A. Firdaus, I. Ghani, and N. I. M. Yasin, "Developing secure websites using feature driven development (fdd): a case study," *Journal of Clean Energy Technologies*, vol. 1, no. 4, pp. 322–326, 2013.

[161] Sonia, A. Singhal, and H. Banati, "Fisa-xp: An agile-based integration of security activities with extreme programming," *ACM SIGSOFT Software Engineering Notes*, vol. 39, no. 3, pp. 1–14, 2014.

[162] C. Pohl and H.-J. Hof, "Secure scrum: Development of secure software with scrum," *arXiv preprint arXiv:1507.02992*, 2015.

[163] D. Baca and K. Petersen, "Countermeasure graphs for software security risk assessment: An action research," *Journal of Systems and Software*, vol. 86, no. 9, pp. 2411–2428, 2013.

[164] OWASP Foundation, "Owasp application security verification standard." https://github.com/OWASP/ASVS.

[165] M. Shore, S. Zeadally, and A. Keshariya, "Zero trust: the what, how, why, and when," *Computer*, vol. 54, no. 11, pp. 26–35, 2021.

[166] Y. Stefinko, A. Piskozub, and R. Banakh, "Manual and automated penetration testing. benefits and drawbacks. modern tendency," in *2016 13th international conference on modern problems of radio engineering, telecommunications and computer science (TCSET)*, pp. 488–491, IEEE, 2016.

[167] W. K. Edwards, E. S. Poole, and J. Stoll, "Security automation considered harmful?," in *Proceedings of the 2007 Workshop on New Security Paradigms*, pp. 33–42, 2008.

[168] B. Johnson, C. Bird, D. Ford, N. Forsgren, and T. Zimmermann, "Make your tools sparkle with trust: The picse framework for trust in software tools," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 409–419, IEEE, 2023.

[169] J. Blythe, "Cyber security in the workplace: Understanding and promoting behaviour change," *Proceedings of CHItaly 2013 Doctoral Consortium*, vol. 1065, pp. 92–101, 2013.

[170] G. Bhandari, A. Naseer, and L. Moonen, "Cvefixes: automated collection of vulnerabilities and their fixes from open-source software," in *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*, pp. 30–39, 2021.

[171] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, "A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each," in *2012 34th International Conference on Software Engineering (ICSE)*, pp. 3–13, IEEE, 2012.

[172] M. Sharma, "Review of the benefits of dast (dynamic application security testing)

versus sast," *International Journal of Management and Engineering Research*, vol. 1, no. 1, pp. 05–08, 2021.

[173] M. Aydos, Ç. Aldan, E. Coşkun, and A. Soydan, "Security testing of web applications: A systematic mapping of the literature," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 9, pp. 6775–6792, 2022.

[174] N. Wilde, B. Eddy, K. Patel, N. Cooper, V. Gamboa, B. Mishra, and K. Shah, "Security for devops deployment processes: Defenses, risks, research directions," *International Journal of Software Engineering & Applications*, vol. 7, no. 6, pp. 01–16, 2016.

[175] A. Thool and C. Brown, "Securing agile: Assessing the impact of security activities on agile development," in *International Workshop on Software Security: Challenges, Opportunities, and Lessons Learned*, 2024.

[176] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin, and X. Hu, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," *ACM Transactions on Knowledge Discovery from Data*, 2023.

[177] H. Jin, Y. Zhang, D. Meng, J. Wang, and J. Tan, "A comprehensive survey on process-oriented automatic text summarization with exploration of llm-based methods," *arXiv preprint arXiv:2403.02901*, 2024.

[178] Y. Fang, Y. Guo, C. Huang, and L. Liu, "Analyzing and identifying data breaches in underground forums," *IEEE Access*, vol. 7, pp. 48770–48777, 2019.

[179] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[180] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[181] S. Mukherjee, A. Mitra, G. Jawahar, S. Agarwal, H. Palangi, and A. Awadallah, "Orca: Progressive learning from complex explanation traces of gpt-4," *arXiv preprint arXiv:2306.02707*, 2023.

[182] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[183] S. Atefi, A. Sivagnanam, A. Ayman, J. Grossklags, and A. Laszka, "The benefits of vulnerability discovery and bug bounty programs: Case studies of chromium and firefox," in *Proceedings of the ACM Web Conference 2023*, pp. 2209–2219, 2023.

[184] M. Blatt and A. Norman, "Email, communication and more: How software engineers use and reflect upon email at the workplace," Master's thesis, 2013.

[185] C. Brown and C. Parnin, "Understanding the impact of github suggested changes on recommendations between developers," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, (New York, NY, USA), p. 1065–1076, Association for Computing Machinery, 2020.

[186] C. Brown and C. Parnin, "Comparing different developer behavior recommendation styles," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ICSEW'20, (New York, NY, USA), p. 78–85, Association for Computing Machinery, 2020.

[187] T. Ahmed and P. Devanbu, "Few-shot training llms for project-specific code-summarization," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1–5, 2022.

[188] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?," in *2013 35th International Conference on Software Engineering (ICSE)*, pp. 672–681, IEEE, 2013.

[189] F. A. F. Limo, D. R. H. Tiza, M. M. Roque, E. E. Herrera, J. P. M. Murillo, J. J. Huallpa, V. A. A. Flores, A. G. R. Castillo, P. F. P. Peña, C. P. M. Carranza, *et al.*, "Personalized tutoring: Chatgpt as a virtual tutor for personalized learning experiences," *Przestrzeń Społeczna (Social Space)*, vol. 23, no. 1, pp. 293–312, 2023.

[190] S. Goyal, E. Rastogi, S. P. Rajagopal, D. Yuan, F. Zhao, J. Chintagunta, G. Naik, and J. Ward, "Healai: A healthcare llm for effective medical documentation," in *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pp. 1167–1168, 2024.

[191] D. Han, T. McInroe, A. Jelley, S. V. Albrecht, P. Bell, and A. Storkey, "Llm-personalize: Aligning llm planners with human preferences via reinforced self-training for housekeeping robots," *arXiv preprint arXiv:2404.14285*, 2024.

[192] M. Jazayeri, "Some trends in web application development," in *Future of Software Engineering (FOSE'07)*, pp. 199–213, IEEE, 2007.

[193] M. Andreessen, "Why software is eating the world," *The Wall Street Journal*, vol. 8, p. 20, 2011.

[194] P. Kumar, P. Sahithi, M. P. Kumar, and B. T. Student, "Implementing scrum and

kanban approaches for e-commerce web application: An agile framework," *vol*, vol. 9, pp. 385–391, 2021.

[195] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017.

[196] A. A. U. Rahman, E. Helms, L. Williams, and C. Parnin, "Synthesizing continuous deployment practices used in software development," in *2015 Agile Conference*, pp. 1–10, IEEE, 2015.

[197] B. Naveen, J. K. Grandhi, K. Lasya, E. M. Reddy, N. Srinivasu, and S. Bulla, "Efficient automation of web application development and deployment using jenkins: A comprehensive ci/cd pipeline for enhanced productivity and quality," in *2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS)*, pp. 751–756, IEEE, 2023.

[198] C. D. Hylender, P. Langlois, A. Pinto, and S. Widup, "2024 data breach investigations report," tech. rep., Verizon Business, 2024. https://www.verizon.com/business/resources/Tac2/reports/2024-dbir-data-breach-investigations-report.pdf.

[199] "New cybersecurity advisory warns about web application vulnerabilities." National Security Agency, 2023. https://www.nsa.gov/Press-Room/Press-Releases-Statements/Press-Release-View/Article/3473830/new-cybersecurity-advisory-warns-about-web-application-vulnerabilities/.

[200] A. Alzahrani, A. Alqazzaz, Y. Zhu, H. Fu, and N. Almashfi, "Web application security tools analysis," in *2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart com-*

puting (hpsc), and ieee international conference on intelligent data and security (ids), pp. 237–242, IEEE, 2017.

[201] K. Petersen, C. Gencel, N. Asghari, D. Baca, and S. Betz, "Action research as a model for industry-academia collaboration in the software engineering context," in *Proceedings of the 2014 international workshop on Long-term industrial collaboration on software engineering*, pp. 55–62, 2014.

[202] M. Staron and M. Staron, "Action research as research methodology in software engineering," *Action Research in Software Engineering: Theory and Applications*, pp. 15–36, 2020.

[203] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, pp. 131–164, 2009.

[204] C. Wohlin, "Case study research in software engineering—it is a case, and it is a study, but is it a case study?," *Information and Software Technology*, vol. 133, p. 106514, 2021.

[205] L. E. Lwakatare, E. Rånge, I. Crnkovic, and J. Bosch, "On the experiences of adopting automated data validation in an industrial machine learning project," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 248–257, IEEE, 2021.

[206] N. Davila, I. Wiese, I. Steinmacher, L. Lucio da Silva, A. Kawamoto, G. J. P. Favaro, and I. Nunes, "An industry case study on adoption of ai-based programming assistants," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, pp. 92–102, 2024.

[207] M. Voggenreiter, F. Angermeir, F. Moyón, U. Schöpp, and P. Bonvin, "Automated

security findings management: A case study in industrial devops," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, pp. 312–322, 2024.

[208] B. A. Myers, "State of the art in user interface software tools," *Readings in Human–Computer Interaction*, pp. 323–343, 1995.

[209] A. M. Memon, "Gui testing: Pitfalls and process," *Computer*, vol. 35, no. 08, pp. 87–88, 2002.

[210] D. Mitropoulos, P. Louridas, M. Polychronakis, and A. D. Keromytis, "Defending against web application attacks: Approaches, challenges and implications," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 2, pp. 188–203, 2017.

[211] M. Luo, O. Starov, N. Honarmand, and N. Nikiforakis, "Hindsight: Understanding the evolution of ui vulnerabilities in mobile browsers," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 149–162, 2017.

[212] M. Stausberg, "Structured observation," *The Routledge Handbook of Research Methods in the Study of Religion*, p. 382, 2011.

[213] J. Fitzpatrick and E. Kostina-Ritchey, "Us families' adoption of chinese daughters: A narrative analysis of family themes in children's books," *Family Relations*, vol. 62, no. 1, pp. 58–71, 2013.

[214] E. Blair, "A reflexive exploration of two qualitative data coding techniques," *Journal of Methods and Measurement in the Social Sciences*, vol. 6, no. 1, pp. 14–29, 2015.

[215] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford, "Questions developers ask while diagnosing potential security vulnerabilities with static analysis," in

Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 248–259, 2015.

[216] D. Oliveira, M. Rosenthal, N. Morin, K.-C. Yeh, J. Cappos, and Y. Zhuang, "It's the psychology stupid: how heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots," in *Proceedings of the 30th Annual Computer Security Applications Conference*, pp. 296–305, 2014.

[217] H. H. AlBreiki and Q. H. Mahmoud, "Evaluation of static analysis tools for software security," in *2014 10th International Conference on Innovations in Information Technology (IIT)*, pp. 93–98, IEEE, 2014.

[218] T. Lopez, H. Sharp, T. Tun, A. Bandara, M. Levine, and B. Nuseibeh, "" hopefully we are mostly secure": Views on secure code in professional practice," in *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pp. 61–68, IEEE, 2019.

[219] A. Poller, L. Kocksch, S. Türpe, F. A. Epp, and K. Kinder-Kurlanda, "Can security become a routine? a study of organizational change in an agile software development group," in *Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing*, pp. 2489–2503, 2017.

[220] H. Alptekin, S. Demir, Ş. Şimşek, and C. Yilmaz, "Towards prioritizing vulnerability testing," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 672–673, IEEE, 2020.

[221] S. Berner, R. Weber, and R. K. Keller, "Observations and lessons learned from automated testing," in *Proceedings of the 27th international conference on Software engineering*, pp. 571–579, 2005.

[222] M. Green and M. Smith, "Developers are not the enemy!: The need for usable security apis," *IEEE Security & Privacy*, vol. 14, no. 5, pp. 40–46, 2016.

[223] F. Kortum, J. Klünder, and K. Schneider, "Behavior-driven dynamics in agile development: The effect of fast feedback on teams," in *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pp. 34–43, IEEE, 2019.

[224] R. Werlinger, K. Hawkey, D. Botta, and K. Beznosov, "Security practitioners in context: Their activities and interactions with other stakeholders within organizations," *International Journal of Human-Computer Studies*, vol. 67, no. 7, pp. 584–606, 2009.

[225] C. Weir, I. Becker, and L. Blair, "Incorporating software security: using developer workshops to engage product managers," *Empirical Software Engineering*, vol. 28, no. 2, p. 21, 2023.

[226] C. Weir, I. Becker, and L. Blair, "A passion for security: Intervening to help software developers," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 21–30, IEEE, 2021.

[227] S. Triantafyllou and C. Georgiadis, "Gamification of moocs and security awareness in corporate training," 2022.

[228] D. P. Gilliam, T. L. Wolfe, J. S. Sherif, and M. Bishop, "Software security checklist for the software life cycle," in *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, pp. 243–248, IEEE, 2003.

[229] J. Martelleur and A. Hamza, "Security tools in devsecops: A systematic literature review," 2022.

[230] A. Heijstek, *Bridging Theory and Practice: Insights into Practical Implementations of Security Practices in Secure DevOps and CI/CD Environments*. PhD thesis, Ph. D. thesis, Universiteit van Amsterdam, 2023.

[231] A. Aggarwal and P. Jalote, "Integrating static and dynamic analysis for detecting vulnerabilities," in *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, vol. 1, pp. 343–350, IEEE, 2006.

[232] S. Al-Amin, N. Ajmeri, H. Du, E. Z. Berglund, and M. P. Singh, "Toward effective adoption of secure software development practices," *Simulation Modelling Practice and Theory*, vol. 85, pp. 33–46, 2018.

[233] V. Akuthota, R. Kasula, S. T. Sumona, M. Mohiuddin, M. T. Reza, and M. M. Rahman, "Vulnerability detection and monitoring using llm," in *2023 IEEE 9th International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE)*, pp. 309–314, IEEE, 2023.

[234] V. Bhasker Reddy Bhimanpati and S. Jain, "Security testing for mobile applications using ai and ml algorithms," *Shalu, Security Testing for Mobile Applications Using AI and ML Algorithms (August 30, 2024)*, 2024.

[235] G. Atluri, "The invisible war on the chain: Cyber defense strategies for enterprise blockchain security," *European Journal of Computer Science and Information Technology*, vol. 13, pp. 112–122, 2025.

[236] A. Thool and C. Brown, "Harnessing the power of llms: Llm summarization for human-centric dast reports," in *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, 2024.

[237] B. Aljedaani, M. A. Babar, *et al.*, "Challenges with developing secure mobile health

applications: systematic review," *JMIR mHealth and uHealth*, vol. 9, no. 6, p. e15654, 2021.

[238] M. Lennartsson, J. Kävrestad, and M. Nohlberg, "Exploring the meaning of usable security–a literature review," *Information & Computer Security*, vol. 29, no. 4, pp. 647–663, 2021.

[239] D. Greer and Y. Hamon, "Agile software development," 2011.

[240] D. S. Cruzes, M. Felderer, T. D. Oyetoyan, M. Gander, and I. Pekaric, "How is security testing done in agile teams? a cross-case analysis of four software teams," in *International Conference on Agile Software Development*, pp. 201–216, Springer International Publishing Cham, 2017.

[241] O. Graham and K. Kloss, "Evaluating the impact of reinforcement learning on autonomous ci/cd workflow optimization," 2025.

[242] V. K. Thatikonda, "Beyond the buzz: A journey through ci/cd principles and best practices," *European Journal of Theoretical and Applied Sciences*, vol. 1, no. 5, pp. 334–340, 2023.

[243] N. Cassee, B. Vasilescu, and A. Serebrenik, "The silent helper: the impact of continuous integration on code reviews," in *2020 IEEE 27th international conference on software analysis, evolution and reengineering (SANER)*, pp. 423–434, IEEE, 2020.

[244] M. I. Lunesu, R. Tonelli, L. Marchesi, and M. Marchesi, "Assessing the risk of software development in agile methodologies using simulation," *IEEE Access*, vol. 9, pp. 134240–134258, 2021.

[245] P. Tominc, D. Oreški, and M. Rožman, "Artificial intelligence and agility-based model

for successful project implementation and company competitiveness," *Information*, vol. 14, no. 6, p. 337, 2023.

[246] M. Zain, R. C. Rose, I. Abdullah, and M. Masrom, "The relationship between information technology acceptance and organizational agility in malaysia," *Information & management*, vol. 42, no. 6, pp. 829–839, 2005.

[247] K. Ramdass and S. Jain, "The role of devsecops in continuous security integration in ci/cd pipe," *Journal of Quantum Science and Technology*, vol. 2, 2025.

[248] J. R. M. Thason and M. Prasad, "Bridging the gap: Integrating security into devops practices for enhanced software delivery and risk mitigation," *International Journal of Research in Modern Engineering Emerging Technology*, vol. 13, pp. 1–22, 2025.

[249] J. C. de Winter, "Controversy in human factors constructs and the explosive use of the nasa-tlx: a measurement perspective," *Cognition, technology & work*, vol. 16, no. 3, pp. 289–297, 2014.

[250] M. A. Akbar, A. A. Khan, S. Mahmood, and S. Hyrynsalmi, "Management of devsecops process: An empirical investigation," *Software: Practice and Experience*, vol. 55, pp. 1234–1255, 2025.