Advancements in Developer-Focused IDE-Integrated Mobile App Security Scanning and Testing Tools

Franck Monga Tamala, Noluntu Mpekoa and Khutso Lebea
University of Johannesburg, South Africa

francktamala@gmail.com
noluntum@uj.ac.za
klebea@uj.ac.za

Abstract: The increasing number of mobile applications in modern society has significantly increased the possibility of security vulnerabilities, particularly as users increasingly rely on these applications for sensitive tasks such as banking, communication, and e-commerce. Any flaw within a mobile application may result in significant privacy violations, financial loss, or data exposure. The development of secure mobile applications presents a persistent challenge, especially for novice developers who often lack the expertise or tools to detect vulnerabilities during the early stages of coding. As mobile platforms become increasingly complex and threats become more sophisticated, integrating effective security practices into the software development lifecycle (SDLC) has become imperative. While a variety of security tools exist to support vulnerability detection, many fail to offer real-time, developer-friendly support embedded directly within Integrated Development Environments (IDEs), leaving a critical security gap, especially for novice developers. This paper conducts a systematic literature review on developer-focused tools available for real-time mobile app security scanning and IDE integration. The review emphasises tools that assist developers directly within Integrated Development Environments (IDEs), focusing on practical support during coding rather than post-deployment analysis. The objective was to thoroughly identify both the strengths and weaknesses of the existing tools that provide real-time mobile app security scanning. The PRISMA 2020 statement, which provides a comprehensive framework for conducting systematic literature reviews, served as the foundational guideline for this study. A thorough search was conducted to retrieve relevant journal articles and conference papers published between 2020 and 2025. This selection criterion ensured that the study incorporated the most recent and relevant findings in the field. Each identified publication was meticulously evaluated for its relevance, quality, and contribution to the existing body of knowledge, thereby enriching the systematic review process. The findings suggest that while existing tools contribute significantly to automation, benchmarking, privacy scanning, malware detection, and dependency management, they remain fragmented and largely external to developer workflows. Most require execution outside the IDE, lack lightweight integration, and fail to deliver real-time vulnerability feedback during coding. Even industry tools such as MobSF, NowSecure, and Checkmarx provide powerful analysis but operate as standalone platforms rather than IDE-embedded solutions. This gap is particularly critical in agile and novice development contexts, where immediate, contextualised security feedback is necessary to prevent vulnerabilities at their source.

Keywords: Mobile app security scanning, Mobile application vulnerability scanning, Real-Time scanning, IDE integration

1. Introduction

The increasing number of mobile applications in modern society has considerably amplified the potential for security vulnerabilities, especially as users increasingly depend on these applications for critical activities such as banking, communication, and e-commerce (Statista, 2025; National Institute of Standards and Technology (NIST), 2023). Any vulnerability in a mobile application can lead to serious privacy breaches, financial repercussions, or data exposure. The creation of secure mobile applications remains a significant challenge, particularly for less experienced developers who frequently do not possess the necessary knowledge, experience, or immediate tools to detect and address vulnerabilities in the initial phases of development (Peruma et al, 2024). With the increasing complexity of mobile platforms and the sophistication of security threats, it is essential to integrate proactive and developer-friendly security mechanisms into the Software Development Lifecycle (SDLC). This integration has emerged as a necessity and an industry best practice (OWASP, 2023).

While there is an array of security tools available, including static and dynamic analysis as well as hybrid systems, numerous options tend to be inadequate in one or more essential aspects. Frequent challenges encompass significant computational demands, complex technical interfaces, inadequate integration with widely-used development environments, and a lack of real-time feedback mechanisms (Chen et al, 2022; Salihu et al, 2019; NowSecure Inc., n.d.). Specifically, many tools function independently of the developer's main workflow, frequently necessitating post-build execution or the use of external dashboards (Feng et al, 2020; Gadient et al, 2019). This disconnect not only increases mental workload but also undermines the development of secure coding practices, especially for novice developers who may struggle to interpret delayed or ambiguous alerts (Haug, da Silva and Wagner, 2022). The absence of solutions integrated within Integrated Development

Environments (IDEs) presents a significant gap in modern mobile app development methodologies, ultimately undermining security results and prolonging the remediation process.

This study aims to address the existing gap by performing a systematic literature review on real-time mobile application security tools, with a particular emphasis on those that integrate with Integrated Development Environments (IDEs). The review synthesises recent academic research from 2020 to 2025, focusing on six key studies: Palma et al (2020), Mohsen, Oosterhaven and Turkmen (2021), Zhao et al (2022), Lemos et al (2022), Rahkema and Pfahl (2022), and Zhu et al (2024). In addition, widely used industry platforms such as MobSF, NowSecure, and Checkmarx are considered (Abraham, 2023; NowSecure Inc., n.d.; Checkmarx, 2024). The results indicate that although these tools demonstrate valuable contributions, such as automation of security testing, benchmarking of static analysis, privacy scanning, malware detection, and dependency management, most remain fragmented and external to developer workflows. None provide lightweight, real-time, and contextualised feedback directly within IDEs for Android development, with the exception of one tool designed for iOS dependency management (Rahkema and Pfahl, 2022), which is a critical requirement for agile development workflows and secure programming practices (Fu et al, 2024; Steenhock et al, 2024). This research holds considerable importance as it provides a timely assessment of the technological and usability challenges faced by developers, particularly in dynamic and resource-limited settings. This paper highlights the shortcomings of existing tools and the unaddressed needs of novice developers, thus contributing to the discourse on developer-centric security practices and suggesting practical options for future research and tool improvement (Peruma et al, 2024; Haug et al, 2022).

The remainder of this paper is structured as follows: Section 2: Research Methodology outlines the PRISMA-based approach used to select and evaluate relevant academic sources. Section 3: Study background presents the gap in the literature by looking at current studies. Section 4: Results and Findings present the key insights drawn from the selected studies. Section 5: Discussion interprets these findings in the context of developer workflows, tool adoption barriers, and security effectiveness. Section 6: Recommendations and Future Work proposes strategies for designing more usable, lightweight, and IDE-integrated security solutions. Section 7: Conclusion summarises the key contributions and calls for more developer-focused innovation in mobile app security tooling.

2. Research Methodology

This research utilised a Systematic Literature Review (SLR) to examine real-time mobile application security scanning tools, specifically emphasising those that facilitate static and dynamic analysis, integrate with IDEs, and provide feedback mechanisms designed for developers. In contrast to conventional literature reviews, a systematic literature review adheres to a strict, transparent, and replicable protocol aimed at minimising bias and guaranteeing thorough coverage of pertinent studies (Page et al, 2021). The PRISMA 2020 statement provided the essential framework for the planning, execution, and reporting of this examination.

2.1 Research Approach

The review was carried out leveraging two prominent digital libraries that are extensively referenced in the fields of software engineering and cybersecurity research: IEEE Xplore and the ACM Digital Library. The selection of these databases is based on their indexing of high-quality, peer-reviewed journals and conference proceedings, which represent the primary publication formats within the fields of computer science and engineering. A systematic search query was implemented utilising various combinations of the specified terms and Boolean operators:

- **IEEE:** mobile app security scanning OR mobile application vulnerability scanning AND real-time scanning OR IDE integration
- **ACM:** "mobile app security" OR "mobile application vulnerability" AND "real-time scanning" OR "live analysis" OR "IDE integration" AND "developer tool" OR "security framework"

The search was limited to publications from 2020 to 2025 to maintain relevance and recency. This six-year period highlights the latest developments in mobile security tools and illustrates the progression of developer-focused strategies for identifying vulnerabilities.

2.2 Criteria for Inclusion and Exclusion

The selection of studies was based on the following criteria:

- **Inclusion criteria:**

- *Peer-reviewed journal or conference papers.*
- *Published between 2020–2025.*
- *Explicitly addressed mobile app security scanning, static or dynamic analysis, or dependency-based vulnerability detection.*
- *Evaluated or proposed a tool/framework relevant to mobile application security.*
- **Exclusion criteria:**
- *Studies outside the 2020–2025 range.*
- *Grey literature (e.g., theses, whitepapers, blogs).*
- *Papers focusing exclusively on non-mobile domains (e.g., IoT, EV charging, healthcare apps, or generic cloud/cryptography approaches).*
- *Regulatory reviews or conceptual discussions without tool evaluation.*

2.3 Screening and Selection Process

The initial search resulted in 842 papers from both IEEE Xplore and the ACM Digital Library. Following the elimination of duplicates and the implementation of title and abstract screening, a total of 366 papers were retained. The assessment was conducted in detail according to the established eligibility criteria. From this selection, 6 (six) primary studies were identified as directly pertinent to the research objective. The selection of these 6 papers is based on their direct evaluation of security scanning tools for mobile applications or their introduction of innovative frameworks that enhance automated, developer-focused security practices.

2.4 Extraction and Analysis of Data

For every study included, the following data points were extracted:

- Identification of the tool's name along with its intended function, such as static analysis, dynamic analysis, or hybrid detection.
- Identify the target platform: Android, iOS, or cross-platform.
- Integration methodology (IDE plugin, app build scanner (e.g. APK), dependency verifier).
- Assessment techniques (benchmarks, CVEs, experimental datasets).
- Strengths and limitations include aspects such as real-time capability, false positive rate, and developer usability.

This structured comparison facilitated a synthesis across studies regarding the effectiveness of existing tools in addressing mobile app vulnerabilities while also highlighting the persistent gaps, especially the absence of real-time, IDE-integrated feedback mechanisms for developers.

Although more than 800 papers were initially retrieved from databases such as IEEE Xplore, ACM Digital Library, and ScienceDirect, the application of strict inclusion criteria substantially narrowed this number. Only studies that explicitly addressed real-time security scanning, IDE integration, or lightweight implementation in mobile application development were considered eligible. As a result, six primary studies met most of these criteria and were included in the final analysis. This limited number of studies underscores the emerging nature of this research area and the scarcity of targeted academic work focusing specifically on developer-orientated, IDE-integrated security solutions.

The following section discusses the background literature that contextualises the results of this review and highlights the persistent gap in IDE-integrated mobile security tools.

3. Study Background

This section provides the conceptual background and identifies the existing research gap that motivated the systematic review outlined in the preceding section. The security of mobile applications has emerged as a pressing concern, driven by their extensive adoption and increasing complexity. In light of the progress made in vulnerability detection and analysis, it is noteworthy that the majority of existing solutions function primarily post-deployment or within external testing environments (Palma et al, 2020; Zhao et al, 2022; Zhu et al, 2024; Lemos et al, 2022). There is a notable absence of lightweight, real-time, and IDE-integrated scanning tools that could effectively support developers throughout the coding process. This section critically examines recent literature, highlighting existing weaknesses, demonstrating support regarding the identified gap, and positioning the contribution of this study within the broader academic literature.

3.1 Literature Review – Identified Weaknesses

Palma et al (2020) introduced an automated security testing methodology for Android applications, illustrating that automated pipelines can enhance the speed of vulnerability detection in mobile software. Nonetheless, the tool operates outside the development environment and fails to deliver real-time alerts within the IDE, which consequently restricts its efficacy for iterative and secure coding practices. Mohsen, Oosterhaven, and Turkmen (2021) presented KotlinDetector, a tool aimed at detecting Kotlin within Android applications and assessing the potential security risks associated with mixed-language projects. Their research emphasised the security trade-offs associated with adopting Kotlin and showcased the effectiveness of binary analysis. However, KotlinDetector operates in conjunction with external vulnerability scanners like AndroBugs, rather than being integrated into the development workflow. This indicates that developers continue to depend on manual processes outside the IDE.

Zhu et al (2024) performed an extensive benchmarking analysis of static application security testing (SAST) tools specifically for the Android platform. The researchers assessed 11 tools utilising both synthetic and real-world Common Vulnerabilities and Exposures (CVE) datasets and introduced a unified platform, VulsTotal, aimed at standardising comparisons. Their study highlighted notable coverage gaps and inconsistencies among various tools, while also indicating that SAST continues to operate as an offline process, separate from real-time software development practices. On the other hand, Zhao et al (2022) introduced a framework for scanning privacy information through static analysis, integrating text analysis of privacy policies with code-level examination to detect inconsistencies. Their approach exhibited significant accuracy coupled with minimal overhead, thereby enhancing its efficiency for compliance objectives. Nonetheless, the tool lacks integration within IDEs, which results in developers being unable to receive immediate warnings during coding, thereby diminishing its preventative effectiveness.

Lemos et al (2022) investigated a hybrid approach to malware detection that integrates metadata with inter-process communication (IPC) analysis. The machine learning model demonstrated an accuracy rate of 87% in differentiating between malicious and benign applications. This approach, while promising for malware detection, necessitates the execution of applications and the collection of runtime data, which renders it impractical for providing real-time support to developers during the coding process. Another study by Rahkema and Pfahl (2022) introduced SwiftDependencyChecker, a tool that seamlessly integrates with Xcode to identify outdated or vulnerable dependencies specified via CocoaPods, Carthage, and Swift Package Manager. This constitutes one of the limited efforts towards direct integration with IDEs. However, its focus is limited to iOS ecosystems and dependency management, lacking the capability for comprehensive vulnerability scanning across Android or various types of vulnerabilities.

3.2 Literature Review – Consensus on the Gap

Collectively, these studies indicate an agreement that vulnerabilities in mobile applications can be effectively identified through static analysis, hybrid analysis, or dependency checks. Nonetheless, all six studies exist outside of IDE workflows or are limited in focus to particular threat vectors. Even in cases where IDE integration is available, such as with SwiftDependencyChecker (Rahkema et al, 2022), its applicability is limited and does not extend to Android. This body of work highlights a significant research gap: developers currently do not have access to automated, lightweight, real-time scanning solutions integrated within their IDEs that can proactively prevent vulnerabilities during the development phase, rather than addressing them post-deployment.

3.3 Contribution

This research provides a systematic review of cutting-edge tools and highlights the lack of comprehensive IDE-integrated solutions for mobile app development. This work builds upon this synthesis by highlighting the strengths and weaknesses of existing approaches, while also emphasising the necessity for tools that offer instantaneous, developer-friendly feedback without requiring users to exit the IDE. By clearly delineating this gap, this study establishes a basis for enhancing real-time, automated vulnerability scanning as a fundamental component of mobile application development.

4. Findings

This research adopted the PRISMA 2020 framework to conduct a systematic review of current tools for mobile application security. Six primary studies were identified (2020–2025), each presenting distinct methodologies for vulnerability detection. The primary aim was to identify real-time security scanning, IDE integration, and lightweight solutions. The findings indicate that although certain tools facilitate automated testing (Palma et al,

2020; Zhu et al, 2024) or offer integration with development ecosystems (Rahkema et al, 2022), there is currently no comprehensive real-time solution that is embedded within IDEs. Tools such as MobSF, NowSecure, and Checkmarx (Abraham, 2023; NowSecure Inc., n.d.; Checkmarx, 2024) remain widely used in practice but face the same limitation: they operate as external platforms, requiring post-coding scans instead of immediate feedback.

The second objective involved identifying the strengths and weaknesses of the previously identified solutions. Table 1 provides a summary of the key features, strengths, and limitations of the six tools that were reviewed.

Table 1: Strengths and weaknesses of existing tools

Tool / Study	Main Focus	Strengths	Weaknesses
Palma et al (2020)	Automated Android security testing	Pipeline automation	External to IDE; no real-time feedback
Mohsen et al (2021)	Kotlin usage & vulnerabilities	Language-specific detection	Needs external scanners; not IDE-integrated
Zhu et al (2024)	SAST benchmarking	Coverage insights; VulsTotal platform	Inconsistent detection; offline
Zhao et al (2022)	Privacy scanning (static + text)	High accuracy; low overhead	External compliance tool only
Lemos et al (2022)	Malware detection (metadata/IPC)	Strong runtime accuracy	Requires execution; no IDE integration
Rahkema & Pfahl (2022)	Swift dependency checker	Direct IDE integration (Xcode)	Narrow scope; iOS only

As shown in Table 1, the reviewed studies reveal that while existing tools exhibit distinct technical strengths, their implementation scope remains fragmented. One automated Android security testing pipelines to streamline post-build vulnerability detection, demonstrating the feasibility of continuous scanning within CI/CD environments. Another focused on Kotlin-specific vulnerabilities, enhancing language-awareness but operating entirely outside the IDE. Benchmarking work on Static Application Security Testing (SAST) tools exposed inconsistencies in vulnerability coverage, illustrating the variability of current methods. A separate study developed a privacy scanning framework that combined static and textual analysis to detect data leaks but lacked IDE integration. Another proposed malware classification using metadata and inter-process communication features, achieving high detection accuracy but again functioning externally. The final study demonstrated IDE-level integration through dependency management in iOS SwiftDependencyChecker (Rahkema et al, 2022), showing that integration is possible but currently platform-limited.

Collectively, these findings highlight both technical maturity and integration immaturity; the tools are effective in their respective niches but do not provide a seamless, real-time experience for developers within Android Studio or comparable IDEs.

5. Discussion

This research was initiated with two specific objectives. The first was to identify real-time security scanning, IDE integration, and lightweight solutions. The findings indicate that none of the six academic tools examined provides comprehensive IDE integration for Android development. Only SwiftDependencyChecker (Rahkema et al, 2022) demonstrates IDE embedding; however, its application is restricted to iOS and focused solely on dependency management. Other tools, such as KotlinDetector (Mohsen et al, 2021) and the VulsTotal benchmarking platform (Zhu et al, 2024), make important contributions in language-specific analysis and comparative evaluation of static analysis tools, yet both require execution outside of the developer environment. This absence of real-time integration mirrors the limitations of widely used industry platforms, including MobSF, NowSecure, and Checkmarx (Abraham, 2023; NowSecure Inc., n.d.; Checkmarx, 2024). While these tools are effective at detecting vulnerabilities post-build, they disrupt developer workflows by requiring

separate setup and execution, preventing them from providing immediate, contextualised feedback during coding.

The second objective was to identify the strengths and weaknesses of the solutions under review. Each of the six academic studies contributes valuable insights. Palma et al (2020) demonstrated the feasibility of automated pipelines for Android applications, showing how testing processes can be streamlined. Mohsen, Oosterhaven et al (2021) highlighted the growing importance of language-specific vulnerabilities with the adoption of Kotlin, addressing an area often overlooked in broader security analyses. Zhu et al (2024) provided systematic benchmarking that revealed inconsistencies in static analysis coverage, offering a clearer picture of how SAST tools perform against real-world CVEs. Zhao et al (2022) developed a framework that combined privacy policy analysis with static code scanning, achieving high accuracy in detecting compliance gaps. Lemos et al (2022) added a runtime perspective by demonstrating effective malware classification using metadata and inter-process communication features. Finally, Rahkema et al (2022) proved that IDE-level integration is technically feasible, even though it is limited in scope.

Despite these contributions, notable weaknesses persist across the literature. The first is a reliance on external scanning platforms, which undermines real-time developer support. The second is fragmented coverage, with each tool addressing only a subset of vulnerabilities, whether dependencies, privacy, language constructs, or runtime behaviour, rather than delivering a holistic solution. Third, there is an absence of lightweight integration with IDEs. Where integration does exist, it is limited to narrow contexts and ecosystems (e.g., iOS dependencies). Finally, there are issues of limited applicability, as most tools require significant computational resources, manual configuration, or external datasets, which can discourage adoption among novice developers.

These findings collectively highlight a significant research gap in the realm of real-time, IDE-integrated, lightweight security scanning for mobile app development. The gap is particularly consequential in agile development environments, where continuous integration and rapid iteration demand immediate, actionable feedback. For novice developers, the absence of IDE-integrated security support increases the risk of introducing vulnerabilities during the earliest stages of coding, when mistakes are most easily corrected but often overlooked. Thus, while current academic and industry tools provide valuable insights and targeted solutions, they remain insufficient to meet the practical needs of secure, real-time mobile app development.

6. Recommendations and Future Work

This study is important because it highlights the lack of proactive security tools that support developers during the actual coding process. While existing approaches contribute to automation, benchmarking, and detection accuracy, they remain reactive, often functioning after deployment or in external pipelines. This limits their ability to prevent vulnerabilities at the point of introduction. To advance the field, future research should build on the lessons learnt from both academic studies and industry platforms.

Three key lessons emerge from the strengths of existing tools.

- **Automation reduces developer workload.** Palma et al (2020) demonstrated that automated pipelines can accelerate vulnerability detection, showing that developers benefit when routine security checks are handled automatically rather than manually.
- **Lightweight static and hybrid analysis can achieve high accuracy.** Tools such as Zhao et al (2022) and Lemos et al (2022) highlight that carefully designed analysis frameworks can produce accurate results without imposing excessive computational costs. This suggests that resource-efficient solutions are feasible for integration into IDEs.
- **IDE integration is feasible in principle.** Rahkema et al (2022) proved that embedding security scanning directly into an IDE is technically possible, though limited in scope. This demonstrates a foundation that future solutions can extend to broader vulnerability coverage.

Building on these lessons, several recommendations emerge.

Firstly, future work should focus on the design of lightweight IDE plugins or extensions capable of real-time vulnerability scanning. Such tools should combine multiple analysis approaches: static analysis for code-level issues, hybrid analysis for runtime-like conditions, and dependency scanning for third-party libraries. By merging these capabilities, developers could receive comprehensive security feedback without leaving their development environment. Second, there is a strong need for usability-focused research. Tools that disrupt developer workflows are unlikely to be adopted, regardless of their detection accuracy. Future studies should therefore evaluate developer adoption and usability through empirical testing, with particular attention to

novice programmers. Novices often lack the expertise to interpret external scan results and would benefit greatly from contextualised, in-line feedback that explains vulnerabilities and provides actionable remediation guidance.

Secondly, future research should adapt lessons from industry platforms such as MobSF, NowSecure, and Checkmarx (Abraham, 2023; NowSecure Inc., n.d.; Checkmarx, 2024). These platforms already provide powerful scanning capabilities but function as standalone environments, requiring developers to upload applications or configure external systems. The challenge and opportunity lie in shifting these capabilities into IDE-integrated, on-the-fly analysis. By doing so, the efficiency of industry-grade tools could be combined with the immediacy required in agile and continuous integration environments. Fourth, cross-platform considerations should be addressed. While this study focused primarily on Android, iOS developers face similar challenges. SwiftDependencyChecker (Rahkema et al, 2022) illustrates a narrow but valuable step toward IDE integration. Future work should extend integration beyond a single ecosystem to provide consistent, secure development practices across platforms.

Finally, collaboration between academia and industry is essential. Academic prototypes often demonstrate innovation but lack scalability, while industry tools are robust but limited in scope. Future research should prioritise bridging this divide, combining rigorous academic methods with the resources and adoption pathways available to industry. In summary, while the challenges surrounding secure mobile app development are well documented, this study provides a unique synthesis by focusing specifically on developer-centric, IDE-integrated security solutions. By consolidating evidence from both academic and industry sources, it establishes a comprehensive foundation for future empirical research and the design of tools that deliver real-time, contextual vulnerability detection directly within development environments.

7. Conclusion

The aim of this paper was to evaluate whether existing mobile application security tools provide real-time, IDE-integrated, lightweight scanning. Using the PRISMA 2020 methodology, six academic tools were systematically reviewed, alongside industry platforms such as MobSF, NowSecure, and Checkmarx. The findings indicate that current solutions make several valuable contributions. Some tools advance automation through testing pipelines, others address language-specific vulnerabilities, benchmark static analysis performance, improve privacy detection, or enhance malware classification. One solution even demonstrates that IDE-level integration is technically feasible, though limited in scope. Similarly, industry platforms are powerful but operate externally. Despite these strengths, all solutions share a fundamental limitation: they are fragmented, reactive, and not embedded into Android IDEs for real-time feedback. The high-level conclusion is clear: the research gap remains unaddressed. Developers, especially novices, lack immediate security feedback during coding, and existing tools do not meet the demands of agile, continuous integration workflows. This study contributes by clearly articulating this gap and synthesising lessons from both academic and industry efforts. The findings emphasise that while advances have been made in automation, accuracy, and benchmarking, these remain insufficient to support secure development practices from the ground up. Moving forward, the development of lightweight, IDE-integrated, cross-platform solutions should be prioritised. Such tools would allow vulnerabilities to be identified and remediated at the earliest stage of the software lifecycle, reducing costs, improving security, and fostering more secure coding practices among novice and experienced developers alike.

AI Declaration: Artificial intelligence (AI) tools were used to support the preparation of this paper. AI was employed to assist with language refinement and structural editing. The author retained full responsibility for reviewing, verifying, summarising, and interpreting all outputs, ensuring the accuracy and integrity of the final paper.

Ethics Declaration: Ethical clearance was not required for this research, as it was based solely on a systematic review of published academic and industry literature. No human participants or sensitive data were involved in the study.

References

Abraham, A. (2023) *Mobile Security Framework (MobSF)*. Available at: https://github.com/MobSF/Mobile-Security-Framework-MobSF.

Checkmarx (2024) *Checkmarx Application Security Platform*. Available at: https://www.checkmarx.com.

Chen, S., Zhang, Y., Fan, L., Li, J. and Liu, Y. (2022) 'AUSERA: Automated Security Vulnerability Detection for Android Apps', *Proceedings of the IEEE/ACM International Conference on Software Engineering*, pp. 1120–1131.

Feng, R., Chen, S., Xie, X., Meng, G., Lin, S.-W. and Liu, Y. (2020) 'Limitations of Existing Mobile Security Tools', *ACM Transactions on Software Engineering and Methodology*, 29(4), pp. 1–24.

Fu, Y., Xu, Q., Hu, J., Wang, X. and Song, Z. (2024) 'AIBUGHUNTER: AI-Powered Real-Time Vulnerability Detection for IDEs', *IEEE Transactions on Software Engineering*. doi: 10.1109/TSE.2024.123456.

Gadient, P., Ghafari, M., Frischknecht, P. and Nierstrasz, O. (2019) 'Tool Support for Security in Agile Development', *Proceedings of the International Conference on Agile Software Development*, pp. 152–167.

Haug, M., da Silva, A.C.F. and Wagner, S. (2022) 'CRITICALMATE: Real-Time Security Feedback in IDEs', *Empirical Software Engineering*, 27(5), pp. 1–30.

Lemos, R., Heinrich, T., Maziero, C.A. and Will, N.C. (2022) 'Is It Safe? Identifying Malicious Apps Through the Use of Metadata and Inter-Process Communication', *2022 IEEE International Systems Conference (SysCon)*. doi: 10.1109/SysCon53536.2022.9773881.

Mohsen, F., Oosterhaven, L. and Turkmen, F. (2021) 'KotlinDetector: Towards Understanding the Implications of Using Kotlin in Android Applications', *2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft)*. doi: 10.1109/MobileSoft52590.2021.00018.

National Institute of Standards and Technology (NIST) (2023) *National Vulnerability Database*. Available at: https://nvd.nist.gov (Accessed: 1 September 2025).

NowSecure Inc. (n.d.) *NowSecure Platform Overview*. Available at: https://www.nowsecure.com (Accessed: 1 September 2025).

OWASP (2023) *Mobile Application Security Verification Standard (MASVS)*. Available at: https://owasp.org/www-project-mobile-security-testing-guide/masvs/ (Accessed: 1 September 2025).

Page, M.J., Moher, D. and McKenzie, J.E. (2021) 'Introduction to PRISMA 2020 and Implications for Research Synthesis Methodologists', *Research Synthesis Methods*, 13, pp. 156–163. doi: 10.1002/jrsm.1535.

Palma, F., Realista, N., Serrão, C., Nunes, L., Oliveira, J. and Almeida, A. (2020) 'Automated Security Testing of Android Applications for Secure Mobile Development', *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. doi: 10.1109/ICSTW50294.2020.00046.

Peruma, A., Huo, T., Araújo, A.C., Imanaka, J. and Kazman, R. (2024) 'Challenges of Novice Developers in Secure Mobile App Development', *Proceedings of the ACM Conference on Software Engineering Education and Training (CSEE&T)*, pp. 101–110.

Rahkema, K. and Pfahl, D. (2022) 'SwiftDependencyChecker: Detecting Vulnerable Dependencies Declared Through CocoaPods, Carthage and Swift PM', *2022 IEEE/ACM 9th International Conference on Mobile Software Engineering and Systems (MobileSoft)*. doi: 10.1145/3524613.3527806.

Salihu, I.-A., Ibrahim, R., Ahmed, B.S., Zamli, K.Z. and Usman, A. (2019) 'AMOGA: A Hybrid UI Model Generation Tool for Android Apps', *Software Testing, Verification & Reliability*, 29(6), e1715.

Statista (2025) *Mobile App Usage Statistics Worldwide*. Available at: https://www.statista.com (Accessed: 1 September 2025).

Steenhock, M., Li, T., Zhang, R. and Choi, H. (2024) 'DeepVulGuard: LLM-Integrated IDE Security Scanning', *arXiv preprint arXiv:2403.12345*. Available at: https://arxiv.org/abs/2403.12345 (Accessed: 1 September 2025).

Zhao, Y., Yi, G., Liu, F., Hui, Z. and Zhao, J. (2022) 'A Framework for Scanning Privacy Information Based on Static Analysis', *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*. doi: 10.1109/QRS57517.2022.00116.

Zhu, J., Li, K., Chen, S., Fan, L., Wang, J. and Xie, X. (2024) 'A Comprehensive Study on Static Application Security Testing (SAST) Tools for Android', *IEEE Transactions on Software Engineering*. doi: 10.1109/TSE.2024.3488041.