IDE Plugins for Secure Android Applications Development: Analysis & Classification Study

Mohammed El Amin TEBIB
Mariem Graa
Oum-El-Kheir Aktouf
Univ. Grenoble Alpes, Grenoble INP, LCIS lab., 26000 Valence,France*
**Institute of Engineering Univ. Grenoble Alpes*
email: mohammed-el-amin.tebib@univ-grenoble-alpes.fr
email: mariem.graa@univ-grenoble-alpes.fr
email: oum-el-kheir.aktouf@univ-grenoble-alpes.fr

Pascal Andre
LS2N
University of Nantes France
email: pascal.andre@ls2n.fr

***Abstract*—In order to increase the security of Android applications, much effort is realised to assist developers in building secure code that is robust against security attacks. In fact, more attention is given to secure the development life-cycle, from requirement analysis to design, coding to test, and every step of the development process. Many security Integrated Development Environment (IDE) plug-ins have been proposed to assist developers in building secure applications. However, as far as we know, there is no study reviewing the existing tools and their effectiveness in detecting known vulnerabilities. The objective of this paper is to close this gap. We developed a classification framework of the current existing security IDE plug-ins in the context of Android application development. This classification framework allows to highlight salient features about 14 selected tools such as: (i) the analysis-based approach, (ii) the vulnerabilities checks coverage, and (iii) the development stage on which these tools could be employed. Obtained results allowed to establish an overview of secure Android applications development. Limits such as: tools unavailability, benchmarks incompleteness, and the need of dynamic analysis approaches are among the significant findings of this study. We believe this work provides useful information for future research on IDE plug-ins for detecting Android related vulnerabilities.**

***Keywords*—Android; Secure Coding; Classification Framework; IDE Plugins.**

I. INTRODUCTION

Mobile applications have become an integral part of our daily life. Android operating systems maintain a leading position with the most significant market share *"70 percent on Feb. 2022"* [1]. In order to address Android users' expectations, the development of Android applications has been growing at a high rate. As a result, Android applications have become an ideal target for attackers to exploit users private data. According to the official MITRE organisation data-source for Android vulnerabilities [2], recent years witnessed the most significant increase of Android security threats, *"1034 vulnerabilities the last couple years"*. And it continues to increase with *"34 vulnerabilities for only the two first months of 2022"*. These vulnerabilities could be exploited to create harmful actions, such as creating malwares and stealing users private information.

In exploratory studies [3][4], Android developers practices are pointed out as the main reason for security vulnerabilities: considering security as a third party activity; lacking awareness about security measures; and making decision in an ad-hoc manner are among the main reasons for considering developers as the first creators of security vulnerabilities. To deal with these issues, both industry and academia have started recently to integrate security into the software development life-cycle, shifting *from* just ensuring the development speed with letting the security checks to external stakeholders, *to* employing new software development paradigms such as **DevSecOps** [5]. In these paradigms, developers are forced to adhere a secure development process by means of training sessions and analysis tools. In this context, it becomes essential to provide Android developers with an overview of existing security analysis plugins. This is the main contribution of our paper. After selecting a sample of open source IDE tools, we proposed a classification framework based on three dimensions: 1) the analysis based approach (static or dynamic); 2) the covered security vulnerabilities by each tool; and 3) the development stage on which these tools could be employed. To limit the scope of our study, the following factors are considered:

- We consider only tools integrated in the IDE environment,
- For industrial tools, we select only free and available ones,
- For academic tools, if the tool is not available, our analysis will be performed through reading the corresponding published paper.

The rest of the paper is organised as follows. Section II introduces material to understand the context and the comparison methodology. Section III summarises the existing related works reviewing the IDE plugins used for securing Android applications development. Section IV presents our proposed classification frameworks. Based on this framework, we present the results of our search and the analysis phases in section V. We give a set of resulting observations in Section VI. Finally, Section VII concludes the paper and provides tracks for future work.

 ISBN: 978-1-68558-007-0

II. BACKGROUND

Android provides a layered software stack composed of native libraries and a framework as an environment for running Android applications. Developers implement different types of applications: (i) **natives**, that restrict their access to Application Programming Interfaces (APIs) provided by the framework and, (ii) **hybrids**, that could also be web applications. Since considering the security of hybrid applications should cover a wide range of potential security issues coming from the web, our study covers only native applications. These applications are built using four types of components: *activities*, *services*, *broadcast receivers* and *content providers*.

Each Android application runs within its own sandbox, which is an isolation mechanism during runtime. Consequently, applications cannot communicate without having proper permissions. Thus, permission system restricts the access to applications, to its components and to system resources (contacts, locations, images, etc) to those having the *required permissions*. Permissions are declared by developers in the manifest file. Their manipulation is shown in many studies as the source of many security issues[6]: privilege escalation resulting from the over declaration of permissions [7], communication issues resulting from the use of undocumented message types of intents [8], etc.

We focus on security vulnerabilities **(Vi)** that could be mistakenly introduced by developers and exploited to craft attacks **(Ai)**. Based on the existing benchmarks such as *Ghera*[9] that contains open source applications implementing vulnerabilities, we started by considering a not exhaustive list of vulnerabilities that belong to the following class of attacks (we intend to extend this list in the future).

1) **A1**. Privilege escalation **(PE)**: this attack occurs when an application with less permissions gains access to the components of a higher privileged application by exploiting one of the following vulnerabilities: *Pending Intent* with empty *base action* (**A1.V1**); Fragments Dynamic Load (**A1.V2**); privileged component export (**A1.V3**); permissions over-privilege (**A1.V4**) or weak permissions checking (**A1.V5**).
2) **A2**. Data Injection: It consists of a malicious manipulation of data to gain control over the system by exploiting Ordered Broadcasts (**A2.V1**); Sticky Broadcasts (**A2.V2**); Components use *call(args)* to invoke provider-defined method (**A2.V3**) or External Storage (**A2.V4**)
3) **A3**. Code Injection: consists of injecting potentially malicious code that is then interpreted/executed by the application using Dynamic code loading without verifying the integrity and authenticity of the loaded code (**A3.V1**)
4) **A4**. Information leaks: they occur when an application private data are accessed by unauthorised applications using Block Cipher algorithm in ECB mode (**A4.V1**) or CBC mode (**A4.V2**) or encryption key stored in the source code (**A4.V3**) or loading files from internal to external storage (**A4.V4**)
5) **A5**. Android components hijack by exploiting Activities that start in a new task (**A5.V1**); Applications with low priority activities (**A5.V2**) or Pending Intent with implicit base intent (**A5.V3**).

Note that for sake of space, more details of each vulnerability are provided in Appendix [10].

The effectiveness of an analysis tool in detecting known vulnerabilities is closely related to the analysis method used by the tool. Analysis approaches are generally classified into 3 groups: (i) Static analysis, which inspects the program without running it to identify coding flaws. It is performed over the Abstract Syntax Tree (AST) that represents the syntax of a programming language as a hierarchical tree-like structure. Furthermore, formal verification can be used to identify errors in code design. (ii) Dynamic analysis, which evaluates the behaviour of the program while it is running based on different methods among them: (1) bytecode Instrumentation to determine how information flows in the program; and (ii) software testing techniques such as Fuzzing to find unknown vulnerabilities. Finally, (iii) Hybrid analysis combines both static and dynamic analysis to improve analysis results.

III. RELATED WORKS

Recent works [11][12] present a general review of existing tools for mobile applications. They list salient features such as their supported IDEs, applicable languages and their abilities to detect security vulnerabilities. However, they do not focus on the Android ecosystem. We focus on Android, and provide a more consistent analysis and finer applicability assets.

Mejía et al. [13] conducted a systematic review to establish the state of the art of secure mobile development. They found seven solutions for assisting secure development. These solutions are classified based on: 1) the type of the use (methodologies, models, standards or strategies); and 2) the related security concern (authentication, authorisation, data storage, data access and data transfer). After analysing the results of this research, we consider that the number of solutions is limited regarding the real existing ones in the literature. In addition, we found that none of the presented solutions is proposed as a tool or a plugin for secure development. In our work, the search and analysis process is more substantial. Indeed, we present a more important number of solutions, which are intended to be used as IDE plugins.

The closest work to our research is the assessment study proposed by Mitra et al. [14]. It evaluates the effectiveness of vulnerability detection tools for Android applications. The authors reviewed 64 tools and empirically evaluated 14 vulnerability detection tools against the *Ghera* benchmark [9] that implements each vulnerability inside a single Android application. As a result, they found that the evaluated tools for Android applications are very limited in their ability to detect known vulnerabilities. The sample of tools in this study is intended for use by pen-testers after the application release. In addition, the evaluation process is limited to the academic tools. In our work, we are interested in academic and industrial free tools, which are specifically designed as security assisting tools.

 ISBN: 978-1-68558-007-0

We did not find existing research work that studies Android IDE plugins from a security perspective. After analysing the existing benchmarks, we consider that *Ghera* repository is the most useful means for evaluating the analysis tools. Indeed, *Ghera* summarises a non-exhaustive list of well known vulnerabilities related to the development of Android applications. It provides an open source Android application implementing each vulnerability. Therefore, to conduct our study, we used the same benchmark as Mitra et al. [14] to evaluate the list of selected plugins.

IV. Classification Framework and Methodology

The classification framework aims to answer the following questions: (i) **Which** IDE plugins are used to check Android application vulnerabilities?; (ii) **How** the selected tools analyse the checked vulnerabilities? (the adopted analysis approach); (iii) **What** are the vulnerabilities among the presented ones are covered by the selected tools?; and finally (iv) **When** these tools can be used in the development process (specification, design, coding and testing). We present in Figure 1 the followed search methodology to identify relevant security assistance tools.

A. Overview of the search methodology

This phase highlights the tools used by designers and/or developers to prevent security issues in Android applications. Our primary source of information were published academic reviews [11] and public GitHub repositories [15] [16]. For industrial plugins, we consider only free and available ones extracted from the OWASP list[17]. We also included the tools we investigated while we were building the PermDroid[18], a tool to prevent permissions related security issues based on formal methods. On the other hand, some excluding criteria are considered: (i) Tools that do not work during the development process like Anadroid [19] (malware detection), and MassVet [20] (analyses packaged applications in Google-Play store); (ii) Tools that cannot be used within the IDE *e.g.* ComDroid [21] warns pen-testers of exploitable inter applications communication errors related to the released applications (see investigations [22][14]); (iii) Tools that are integrated in the IDE but are not concerned by security vulnerabilities, like PMD [23]. These tools are used for checking coding standards, class design problems, but cannot be used for identifying code smells related to security issues.

B. Shallow analysis

The analysis process here is performed through only reading the available documentation and/or the published corresponding papers. We dug the documentations on many stages. Some vulnerabilities such as **A1.V4** (the over-privilege use of permissions) have been investigated by some of our students and revised by the first author of this paper. The remaining vulnerabilities analysis is realised by the first author and revised by the second author. Other features relevant to our study are also extracted.

C. Deep analysis

In this phase we perform an experimental analysis that completes the preceding one. It consists of performing an empirical evaluation by *running* the selected tools against the defined vulnerabilities according to the evaluation process summarised in Figure 2.

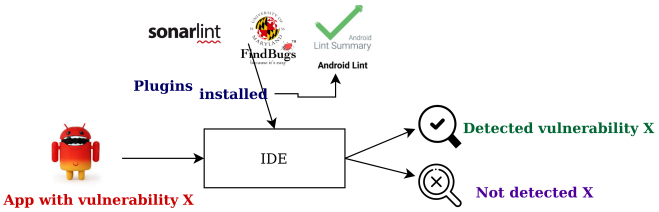

Figure 2. Deep Analysis Process

This evaluation is conducted for only available and free tools such as Sonarlint, Androidlint, FixDroid and FindBugs. We attempted to experiment more tools but this was not possible due to the unavailability of the tools. We contacted the authors of PerHelper, 9Fix and Vandroid but we did not receive an answer yet. Consequently, we decided to perform a second iteration on the documentation analysis for the unavailable tools instead of experimenting them (which was not possible). Finally, as our study is on vulnerabilities that could be found at the code level, our deep analysis could not be applied on tools such as Sema, PoliDroid–As, Page because the inputs of these tools are respectively: GUI Storyboards for Sema, Textual specification for PoliDroid–As and Page of the application, and not the application source code.

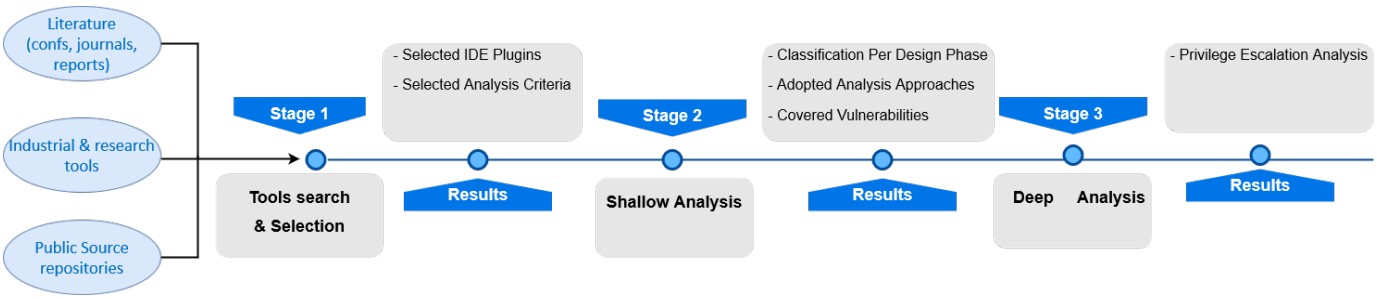

Figure 1. Search Methodology

TABLE I
IDE SECURITY ANALYSIS TOOLS FOR ANDROID APPLICATIONS

Tool Name	Ref.	Year	SD Stage	Focus	Approach	Method	Availability	AV
Curbing	[7]	2011	CR	Permission Over-privilege	Static, Manual	AST	No	2.2
Lintent	[24]	2013	CR	Communication	static	FM	Ye	4.x
PermitMe	[25]	2014	CR	Permission Over-privilege	Static	AST	No	5.0
Page	[26]	2014	Spec	Privacy policies	Static	NL	No	-
Vandroid	[27]	2018	CR	Communication	Static	FM	No	9.0
Androidlint	[28]	2019	CR	Communication	Static	AST	Yes	all
Sema	[29]	2019	Design	General Security Properties	Static	FM	Yes	10
PerHelper	[30]	2019	CR	Permission Over-privilege	Static	AST	No	10
PoliDroid-As	[31]	2017	Spec	Privacy security policies	Static	NLP	No	8
9Fix	[32]	2021	CR	General Code smells	Static	AST	No	12
Sonarlint	[33]	2021	CR	General Code smells	Static	TA	Yes	12
FindBugs	[34]	2016	CR	General Code smells	Static	AST	Yes	7
Cocunut	[35]	2018	Spec	Privacy policies	Static	H	Yes	-
FixDroid	[36]	2017	CR	General Code smells	Static	AST	Yes	7

[1] AST: Abstract Syntax Tree; CR: Code Review; FM: Formal Methods; Spec: Specification;
[2] SD Stage: Software Development Stage; AV: Android Version

V. ANALYSIS AND EVALUATION RESULTS

In this Section, we present analysis results of tools with regards to the classification criteria presented in Section IV.

A. Shallow analysis

1) IDE plugins: As a result of the applied selection process. We present 14 plugin for Android vulnerabilities analysis (*cf.* Table I). We reported for each tool the software development stage (SD stage), the type of covered security vulnerabilities (Focus), the analysis approach, the analysis method, its availability and the Android Version (AV), a useful up-to-date information.

2) Analysis approaches: 98% of the analysis approaches are static, mainly AST analysis and formal methods.

- **Static AST Analysis** Most of IDE plugins investigate statically the program AST provided by the IDE such as Sonarlint , FindBugs and AndroidLint. Other tools, such as PerHelper, PermitMe and Curbing also investigate the AST to find the declared permissions in the application and the list of API calls requiring those permissions. The goal is to detect extra declared permissions that are not associated to any API call.
- **Formal Methods**: Lintent analyses the data-flow to formally check flow information with regards to security properties. Lintent uses the Formal Calculus for reasoning on the Android inter-component communication API, and Type and Effect to statically prevent privilege escalation attacks on well typed components. In the same line, Sema uses Formal Verification of security properties in order to generate a secure code.

When comparing our observations with the security analysis methods presented in Section II, we found that only some static ones are adopted by studied plugins. Dynamic and hybrid approaches are not referred despite their advantages. We underline this point in detail in Section VI.

3) Security vulnerabilities: The shallow analysis covers all the vulnerabilities of Appendix [10]. For each category, we observe whether the associated vulnerabilities are covered (or not) by the tools. We consider True Positive [TP] (resp. False Negative [FN]) cases: a vulnerability is present and detected (resp. not detected) by the tool. We identified three main classes:

- Tools that are specialised in a specific and unique security concern were *easy* to investigate. Based on the corresponding published papers for the plugins: Curbing, PermitMe and PerHelper. They are clearly specialised in detecting privilege escalation attacks **(A1)** resulting from the extra use of permissions **(V4)** in the application. For other tools such as 9Fix, the list of covered vulnerabilities was explicitly declared in the related paper. Consequently, it was easy to know that these tools detect **A3.V1** vulnerability. Last but not least, Coconut, FindBug, Page and PoliDroid−As cover other types of vulnerabilities not included in our study.
- Tools specialised in detecting a specific type of attacks but the number of covered vulnerabilities is too large are *less easy* to investigate. As an example, Lintent could theoretically detect a large number of vulnerabilities as it formalises a notion of safety against privilege escalation. Based on the related published paper, it was not easy to decide whether the tool detects the vulnerability or not as the described formal model was too general. Fortunately, we found the list of covered vulnerabilities mentioned in the corresponding git repository [37]. Thus, we found that **A1.V1**, **A1.V4** and **A5.V4** are covered by the tool. For Sema it is explicitly declared that it covers all the vulnerabilities present in *Ghera*. However, we could not experiment the tool as the inputs of Sema are graphical storyboards and not source code.
- Finally, for industrial tools such as Androidlint , Sonarlint , FixDroid, it was hard to investigate the covered vulnerabilities based on the documentation. The scope of these tools is general and the documentation is too large. We found that the following vulnerabilities: **A2.V1**, **A4.V1, A4.V2, A4.V3** are covered by Sonarlint . For the remaining properties, we did not found any information

 ISBN: 978-1-68558-007-0

indicating whether they are covered by these tools or not.

4) Design Level: It is broadly admitted that security concerns should be handled as early as possible during the development. Secure development lifecycle (SDLC) methodologies have been adopted by many software organisations, e.g., Microsoft through their Microsoft Security Development Lifecycle (SDL) [38], OWASP with their SDLC and Software Assurance Maturity Model (SAMM) processes [39], etc. Table I shows that most tools focus on coding:

- *Specification*: PoliDoid−AS, Page, Cocunut
- *Design*: Sema, Vandroid
- *Coding & Testing*: Curbing, Lintent , PermitMe, Androidlint, Vandroid, 9Fix, PerHelper, Sonarlint , FindBugs, FixDroid.

On the one hand, we found that most of IDE plugins are considered at the *coding* phase of the development life cycle. They act as code review tools notifying developers about their ”unconscious” security issues. On the other hand, a few works allowing security checks at *specification*, *design* and *verification* phases have been proposed.

B. Deep analysis results

The objective of this part of our study is to confirm shallow analysis results with an experimental evaluation using Android application benchmarks. We mainly focused on the considered vulnerabilities, especially the A1 (privilege escalation) attacks. Indeed, we found in CVE details [2], that privilege escalation witnessed the most significant increase among the Android security threats in the last couple years. Vulnerabilities related to Privilege escalation also represent 69.9% of attacks against Android applications. The results are published in the *technical report* [10].

We can observe that the deep evaluation confirmed that the following tools: Curbing, PermitMe, and PerHelper are specifically oriented to detect over-privilege vulnerabilities (A1.V4) and not the other vulnerabilities (A1.V1, A1.V2, A1.V3, A1.V5). Our deep evaluation also confirmed that none of the privilege escalation vulnerabilities are covered by Sonarlint , FindBugs, FixDroid and Androidlint . For tools that are not available (Vandroid and 9Fix), an additional careful documentation-based analysis also confirmed that none of the privilege escalation vulnerabilities is covered.

To conclude, none of the studied tools covers all the privilege escalation attacks and we plan to tackle this limitation.

VI. Discussion

Our analysis study raised some lessons:

- Tools outdatedness and availability: Since the creation of the first version of Android in 2008, the system and the framework levels have shown many security improvements to protect users privacy. A new Android version is released every six months. As a consequence, most of the security assisting IDE plugins become outdated, and not able to deal anymore with new types of application components, or new released APIs. Four factors are of interest when considering outdated tools: (i) the date of the last commit, (ii) the supported IDE type, (iii) information leaks, (iv) the integration of the tools within the last IDE versions. Besides observing that the date of the last commit for many tools is old, most tools are still supported by Eclipse only, which is no more used for developing Android applications. Furthermore, among the proposed tools, only a few is available for use in real Android development projects. Hence, among the 14 analysed tools, eight academic tools are not available for use.
- Tools Effectiveness: Tools such as Lintent , PerHelper, PermitMe are based on Felt et al. [40] permission mapping over-privileged applications detection. This permission mapping is outdated and does not consider an accurate permission set. Our study shows that none of the assessed industrial plugin covers over-privilege vulnerabilities.
- Analysis approaches for security: as observed in Section II, most tools use static approaches to extract information that enables to check the validity of security properties patterns. As a first direction of improvement, static analysis performances of IDE plugin could be improved by adopting complementary analysis techniques such as Symbolic Execution, to allow sound results in case of inter-component communication analysis. Other static analysis techniques have started being used by static analysis tools like SonarQube. The latter tool performs Static Taint Analysis to detect vulnerabilities related to fault injection. Finally, we were surprised to observe that none of the investigated tools takes advantage from the integrated IDE Android simulator to perform dynamic analysis. Adopting dynamic analysis approaches could be an interesting direction to improve security IDE plugin analysis results. This enables to analyse API calls performed dynamically. Furthermore, other dynamic analysis techniques could be used such as dynamic code instrumentation to exploit run-time source code, and fuzzing as a software testing technique for automatic input generation.
- Benchmark availability and incompleteness: *Ghera* is an excellent reference to be used to evaluate the security analysis plugins that deal with open source projects, as it implements an open source application with most known vulnerabilities. However, it suffers from the lack of some vulnerabilities, such as service hijack. It also suffers from the lack of a complete description (component hijacking description). Availability of more relevant benchmarks could be a real breakthrough towards more thorough security analysis.

VII. Conclusion and Future Work

In order to secure Android applications development against software vulnerabilities, it is necessary to integrate security in the software development cycle for assisting developers. In this paper, we provided Android developers an overview of existing security analysis plugins capabilities with regards to Android application development. To provide meaningful and exploitable results, we performed two types of analysis: a

 ISBN: 978-1-68558-007-0

shallow analysis, then an experimental analysis for evaluating the selected IDE plugins security coverage against the defined vulnerabilities. In the empirical part of our study, we mainly focused our efforts on privilege escalation vulnerabilities as these ones are among the hardest vulnerabilities to mitigate, and are related to a complementary research work within our team. Our study highlighted two main research gaps, which could benefit from future work such as: the need of developing tools that cover the whole life-cycle; and enrich the existing benchmarks by new open source applications implementing other Android related vulnerabilities.

The main perspectives related to our work will consider: 1) Extending the list of analysed vulnerabilities to better cover the presented attacks; 2) adding new attacks related to networking, web and phishing; 3) and completing the empirical analysis step.

REFERENCES

[1] Global market share held by mobile operating systems since 2009. [Online]. Available: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

[2] Cve details: Android vulnerability statistics. Retrv: 12, 2021. [Online]. Available: https://www.cvedetails.com/product/19997/

[3] R. Balebako, A. Marsh, J. Lin, J. I. Hong, and L. Cranor, "The Privacy and Security Behaviors of Smartphone App Developers," in *USEC Workshop, NDSS 2014*. The Internet Society, 2014.

[4] G. L. Scoccia, A. Peruma, V. Pujols, I. Malavolta, and D. E. Krutz, "Permission issues in open-source android apps: An exploratory study," in *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2019, pp. 238–249.

[5] Z. Ahmed and S. C. Francis, "Integrating security with devsecops: Techniques and challenges," in *2019 International Conference on Digitization (ICD)*. IEEE, 2019, pp. 178–182.

[6] A. K. Jha, S. Lee, and W. J. Lee, "Developer mistakes in writing android manifests: An empirical study of configuration errors," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 25–36.

[7] T. Vidas, N. Christin, and L. Cranor, "Curbing android permission creep," in *Proceedings of the Web 2.0 Security and Privacy 2011 workshop (W2SP 2011)*, 2021.

[8] W. Ahmad, C. Kästner, J. Sunshine, and J. Aldrich, "Inter-app communication in android: Developer challenges," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016, pp. 177–188.

[9] Ghera repository. Retrv: 07, 2022. [Online]. Available: https://bitbucket.org/secure-it-i/android-app-vulnerability-benchmarks/

[10] "Ide plugins evaluation against privileges escalation attacks," TR-2022. [Online]. Available: https://uncloud.univ-nantes.fr/index.php/s/mzwoC44xs5xiowN

[11] J. Li, S. Beba, and M. M. Karlsen, "Evaluation of open-source ide plugins for detecting security vulnerabilities," in *Proceedings of the Evaluation and Assessment on Software Engineering*, 2019, pp. 200–209.

[12] A. Z. Baset and T. Denning, "Ide plugins for detecting input-validation vulnerabilities," in *2017 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2017, pp. 143–146.

[13] J. Mejía, P. Maciel, M. Muñoz, and Y. Quiñonez, "Frameworks to develop secure mobile applications: A systematic literature review," in *World Conference on Information Systems and Technologies*. Springer, 2020, pp. 137–146.

[14] V.-P. Ranganath and J. Mitra, "Are free android app security analysis tools effective in detecting known vulnerabilities?" *Empirical Software Engineering*, vol. 25, no. 1, pp. 178–219, 2020.

[15] Android references. Retrv: 03, 2022. [Online]. Available: https://github.com/impillar/AndroidReferences

[16] Android security assessment tools. Retrv: 09, 2021. [Online]. Available: https://bitbucket.org/secure-it-i/android-app-vulnerability-benchmarks/

[17] Owasp - source code analysis tools. Retrv: 03, 2022. [Online]. Available: https://owasp.org/www-community/Source_Code_Analysis_Tools

[18] M. E. A. Tebib, P. André, O.-E.-K. Aktouf, and M. Graa, "Assisting developers in preventing permissions related security issues in android applications," in *European Dependable Computing Conference*. Springer, 2021, pp. 132–143.

[19] S. Liang, A. W. Keep, M. Might, S. Lyde, T. Gilray, P. Aldous, and D. Van Horn, "Sound and precise malware analysis for android via pushdown reachability and entry-point saturation," in *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*, 2013, pp. 21–32.

[20] K. Chen, P. Wang, Y. Lee, X. Wang, N. Zhang, H. Huang, W. Zou, and P. Liu, "Finding unknown malice in 10 seconds: Mass vetting for new threats at the {Google-Play} scale," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 659–674.

[21] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, 2011, pp. 239–252.

[22] I. Ul Haq and T. A. Khan, "Penetration frameworks and development issues in secure mobile application development: A systematic literature review," *IEEE Access*, 2021.

[23] Pmd-idea. Retrv: 03, 2022. [Online]. Available: https://plugins.jetbrains.com/plugin/4596-qaplug--pmd

[24] M. Bugliesi, S. Calzavara, and A. Spanò, "Lintent: Towards security type-checking of android applications," in *Formal techniques for distributed systems*. Springer, 2013, pp. 289–304.

[25] E. Bello-Ogunu and M. Shehab, "Permitme: integrating android permissioning support in the ide," in *Proceedings of the 2014 Workshop on Eclipse Technology eXchange*, 2014, pp. 15–20.

[26] M. Rowan and J. Dehlinger, "Encouraging privacy by design concepts with privacy policy auto-generation in eclipse (page)," in *Proceedings of the 2014 Workshop on Eclipse Technology eXchange*, 2014, pp. 9–14.

[27] A. Nirumand, B. Zamani, and B. T. Ladani, "Vandroid: A framework for vulnerability analysis of android applications using a model-driven reverse engineering technique," *Softw. Pract. Exp.*, vol. 49, no. 1, pp. 70–99, 2019. [Online]. Available: https://doi.org/10.1002/spe.2643

[28] Improve your code with lint checks. Retrv: 03, 2022. [Online]. Available: https://developer.android.com/studio/write/lint

[29] J. Mitra, V.-P. Ranganath, T. Amtoft, and M. Higgins, "Sema: Extending and analyzing storyboards to develop secure android apps," *arXiv preprint arXiv:2001.10052*, 2020.

[30] G. Xu, S. Xu, C. Gao, B. Wang, and G. Xu, "Perhelper: Helping developers make better decisions on permission uses in android apps," *Applied Sciences*, vol. 9, no. 18, p. 3699, 2019.

[31] R. Slavin, X. Wang, M. B. Hosseini, J. Hester, R. Krishnan, J. Bhatia, T. D. Breaux, and J. Niu, "Toward a framework for detecting privacy policy violations in android application code," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 25–36.

[32] A.-D. Tran, M.-Q. Nguyen, G.-H. Phan, and M.-T. Tran, "Security issues in android application development and plug-in for android studio to support secure programming," in *International Conference on Future Data and Security Engineering*. Springer, 2021, pp. 105–122.

[33] Sonarlint ide extension for code security. [Online]. Available: https://www.sonarlint.org/

[34] Findbugs-idea. Retrv: 02, 2022. [Online]. Available: https://plugins.jetbrains.com/plugin/3847-findbugs-idea

[35] T. Li, Y. Agarwal, and J. I. Hong, "Coconut: An ide plugin for developing privacy-friendly apps," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 4, pp. 1–35, 2018.

[36] D. C. Nguyen, D. Wermke, Y. Acar, M. Backes, C. Weir, and S. Fahl, "A stitch in time: Supporting android developers in writingsecure code," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1065–1077.

[37] Lintent: Towards security type-checking of android applications. Retrv: 03, 2022. [Online]. Available: https://github.com/alvisespano/Lintent

[38] Explore the microsoft security sdl practices. Retrv: 04, 2022. [Online]. Available: https://www.microsoft.com/en-us/securityengineering/sdl

[39] Software assurance maturity model. Retrv: 04, 2022. [Online]. Available: https://owasp.org/www-project-samm/

[40] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 627–638.

 ISBN: 978-1-68558-007-0