

ArmorDroid: A Rule-Set Customizable Plugin for Secure Android Application Development

Cong-Binh Le^{1,2}, Bao-Thi Nguyen-Le^{1,2}, Phuoc-Loc Truong^{1,2}, Minh-Triet Tran^{1,2}
and Anh-Duy Tran^{3,*}

¹Faculty of Information Technology, University of Science, Ho Chi Minh City, Vietnam

²Vietnam National University, Ho Chi Minh City, Vietnam

³imec-DistriNet, KU Leuven, Leuven, Belgium

fi

Keywords: Android Studio Plugin, Security Coding, Rule-Set Customizable, DevSecOps.

Abstract: Although Android is a popular mobile operating system, its app ecosystem could be safer. The lack of awareness and concern for security issues in apps is one of the main reasons for this. Given the current situation, developers have yet to receive sufficient security knowledge. Therefore, we have researched and proposed a tool to support security coding. Based on the idea of DevSecOps, developers are placed at the center to optimize the solution to this problem by integrating security programming into the earlier stage in the software development process. This paper presents two main research contributions: compilation and categorization of security issues in Android application development and developing ArmorDroid, a plugin for Android Studio to support secure coding. This plugin, which can be used for Java, Kotlin, and XML files, can instantly scan and detect vulnerable code and suggest quick fixes for developers during the development phase. This plugin helps developers improve their security code and trains them to write secure code by providing security coding standards in Android applications. Furthermore, developers can customize our rule set to suit their situation and share it with different developers. Our work also presents the results of a pilot study on the effectiveness of the ArmorDroid plugin.


1 INTRODUCTION


Smartphones are being used more and more frequently and are crucial in many facets of life. The two leading operating systems for smartphones are Android and iOS. According to Statista (Statista, 2023), in the fourth quarter of 2022, approximately 27.5 billion apps were downloaded from Google Play, while the Apple App Store had roughly 8.1 billion downloads. Android apps have a shorter publishing time than iOS apps. However, this also means that the Google Play Market has more malicious apps than the Apple App Store, which has stricter quality control. In many cases, high-ranking apps with hundreds of thousands of downloads from the Google


Play Store have been discovered to contain security issues (Faruki et al., 2014).


The primary responsibility for security problems in Android apps remains a topic of debate. According to the findings of the GitLab DevSecOps Survey 2022 (Gitlab, 2022), many security experts lack confidence in developers' ability to write secure code, while developers feel they lack guidance. In their study, Tran *et al.* (Tran et al., 2021) proposed that the main causes of the above issues are a lack of secure coding standards for mobile software development, varying levels of developer knowledge and qualification in secure coding practices, a focus on functionality, performance, and deadlines over security, and the skipping of important security steps to meet tight deadlines. The reasons above suggest that various stakeholders have not prioritized security issues in Android apps, including software development companies involved in the development process.


DevSecOps is a practice that integrates security testing throughout the software development lifecycle. It aims to "shift-left" security, meaning to involve

^a <https://orcid.org/0009-0009-8546-5440>

^b <https://orcid.org/0009-0000-3371-6986>

^c <https://orcid.org/0009-0008-5665-9986>

^d <https://orcid.org/0000-0003-3046-3041>

^e <https://orcid.org/0000-0002-8036-954X>

*Corresponding author

security from the initial design phase to the final delivery phase.

Mitigating common vulnerabilities requires secure coding practices to be integrated into the earliest stages of development. Therefore, a tool to help programmers track and fix insecure code right in the coding process is necessary. Several plugins have been developed to support secure coding based on the mentioned idea. However, we have identified some limitations with using these plugins from both commercial and open source. For example, FixDroid (Nguyen et al., 2017) cannot extend the rules or add more tooltips for new or special security issues. 9Fix (Tran et al., 2021) can only support Java programming language and run on Android Studio versions earlier than 3.0.

In this work, we propose a new Android Studio plugin that supports secure programming for developers, called **ArmorDroid**. It can detect and fix common security issues in Java, Kotlin, and XML files. It works with new versions of Android Studio and runs locally on the developer's machine to ensure privacy. In addition, the tool provides a rule management system and support for rules customization. In this paper, our main contributions are:

- Identifying and categorizing security issues in Android applications. (Section 3)
- Building a plugin for Android Studio to support secure coding, ArmorDroid. Designing a data model that generalizes security issues collected into a set of rules. (Section 4)
- Running the pilot study and analyzing the results for ArmorDroid. (Section 5)

2 RELATED WORK

DevSecOps is a practice that ensures security throughout the software development lifecycle. It involves applying security principles and tools from the design phase to the delivery phase. For Android development, there are various security testing tools that can be integrated into the CI/CD pipeline such as AppSweep (GuardSquare, 2023), App-Ray (App-Ray, 2023), and MobSF (mob, 2023). These tools can perform both Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) of the source code and detect vulnerabilities. However, they can only be used when code is ready for testing.

By performing static code analysis, we can inspect the control-flow and the data-flow of the program without executing it. EstiDroid is an open-source tool

developed by Fan *et al.* (Fan et al., 2020). It analyzes the API calls of Android apps by scanning the APK file. This method is faster than dynamic analysis but less accurate. Some solutions create IDE plugins that catch insecure code right at the coding time. SonarLint (son, 2023b) is a commercial product consisting of plugins for various IDEs and languages, such as Python, JavaScript, and Java. Another example is DroidPatrol by Talukder *et al.* (Talukder et al., 2019). It is a plugin for Android Studio that uses static analysis to find data leaks in APK files. However, this plugin only works after the code is compiled, not during coding. 9Fix by Tran *et al.* (Tran et al., 2021) and Sensei by Cremer *et al.* (De Cremer et al., 2020) are two plugins that help programmers write secure code and follow coding guidelines. They provide real-time feedback and suggestions to fix vulnerabilities and improve code quality. A current drawback of these two plugins is that they do not support Kotlin, which is a modern and popular language for Android app development.

3 SECURITY ISSUES IN ANDROID APPLICATION

This section covers various security problems that affect Android apps and how they occur. This helps us learn from the common mistakes of Android developers and find a way to fix them.

3.1 Security Issues Related to Cryptography Implementation

Cryptography is essential for securing data in transit and at rest. Cryptography security issues can lead to data leaks and compromise the confidentiality, integrity, and availability of sensitive data and communications. Cryptography vulnerabilities make it easy for attackers to steal data and violate privacy (OWASP, 2016). To avoid security issues in cryptography implementation in Android applications, developers should follow the standard guidelines for cryptography algorithms and data protection.

3.2 Security Issues in Network Communications

Security issues in Network communication in Android applications refer to vulnerabilities or weaknesses in how an application interacts with other devices or servers via a network. These problems can expose the application and the data it contains

to attack, increasing the risk of data breaches, system compromises, or other security incidents. Some common security issues include Man-in-the-Middle (MITM) attacks, insecure communication protocols, a lack of authentication and access control, etc.

3.3 Security Issues in Data Storage

This category refers to security issues in the way that an application stores and manages user data, which can leave the application and its data open to attack. SQL Injection is the most common type of security issue. An attacker can inject malicious SQL commands into a database management system of an application to gain unauthorized access to or edit sensitive data.

3.4 Security Issues in Interprocess Communication

The Android Interprocess Communication (IPC) mechanisms let you verify the identity of the application connecting to your IPC and set security policy for each IPC mechanism such as Intent, Binder, Messenger with a Service, and BroadcastReceiver. Interprocess communication issues can lead to unauthorized access to data and code execution.

3.5 Security Issues in WebView

WebView in Android allows applications to display web content (HTML, CSS, JavaScript). While this feature is very useful for many cases, improper use of it can introduce web security issues into your app, such as injection attacks, MITM attacks, clickjacking, and cross-site scripting (XSS) attacks. A mobile application that uses WebViews can have XSS issue if untrusted code is executed with local file access enabled.

4 SECURE CODING PLUGIN FOR ANDROID STUDIO

Android Studio (Google, 2023) is an IDE for developing Android apps. It is based on IntelliJ IDEA Platform, a code editor and developer toolset. It provides app development features like code completion, debugging, testing, and deployment. Moreover, third-party plugins can be written to add new features to the IDE.

4.1 Overview of ArmorDroid

Android Studio allows several types of extensions, two of which are suitable for the tool our team is planning to develop. Firstly, the **User Interface Components** provide the ability to add some UI features to the IDE such as MenuBar, ToolBar, etc. Secondly, the **Code Analyzing** feature provides syntax highlighting, language checking, source code inspection, etc., which allows our plugin to check for vulnerabilities. Using the available extension features, we have implemented a secure coding plugin for Android Studio, named **ArmorDroid**. Our plugin’s architecture is described in Figure 1.

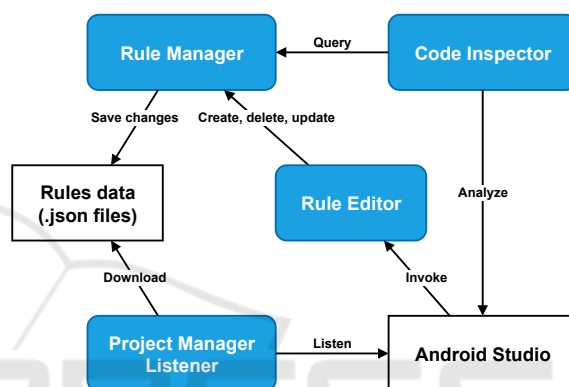


Figure 1: Architecture of ArmorDroid.

When the user opens a project, the *Project Manager Listener* will prompt a dialog asking for user permission to inspect this project. If the user agrees, the plugin will create a folder called *ArmorDroidRule-Data* and download the default rule-storage files into that folder: *java.json*, *kotlin.json* and *xml.json*. The downloaded files are used locally by the *Code Inspector* module. No data are transmitted to any web server to ensure the confidentiality of the project source code. The *Rule Editor* module includes actions and dialogues that are responsible for interacting with the user. To modify the rules, *Rule Editor* calls APIs provided by the *Rule Manager* module. *Rule Manager* is responsible for interacting with the rule files and providing the right set of rules to *Code Inspector* according to the language they are scanning. Every time changes are made to the rule files, *Rule Editor* will notify the *Code Inspector* to refresh the data.

4.2 Design and Implementation

4.2.1 Code Inspector

To check the rules, our tool utilizes the IDE syntax parsing features called Program Structure Interface

```
//create the cipher from the passed in type using the specified security provider
Cipher cipher = Cipher.getInstance(transformation: "AES/CBC/PKCS5Padding", SECURITY_PROVIDER);
//generate
SecretKey : ArmorDroid: Default ECB mode should not be used
//init the
cipher.ini ArmorDroid: Improve security by using AES/GCM/NoPadding Alt+Shift+Enter More actions... Alt+En
//create the cipher outputStream
Fetching Documentation...
cos = new CipherOutputStream(outData, cipher);
```

Figure 2: Highlighted Vulnerable Code.

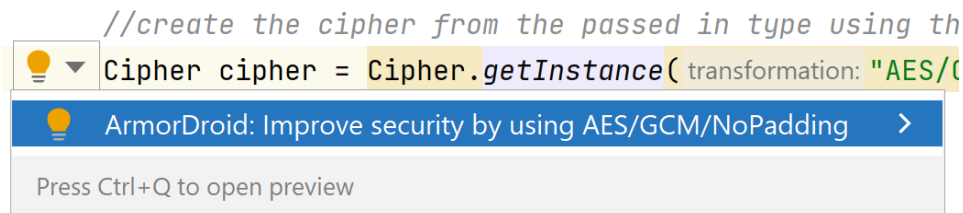


Figure 3: Quick Fix Menu.

(PSI). The PSI is the layer in the IntelliJ Platform responsible for parsing files and creating the syntactic and semantic code model. When a developer writes new code, the IDE rebuilds the Abstract Syntax Tree (AST) and computes the changes compared to the previous version. In simple terms, the AST breaks the source code into smaller components such as *PsiElement*, *PsiWhiteSpace*, *PsiClass*, *PsiMethod*, *PsiField*, etc. A typical PSI element will contain the following information: source text, its parent, its children, and its position in the source file. Since we aim to detect and highlight problematic code as soon as the developer writes it, the *localInspection* (loc, 2023) configuration was chosen for our inspection method.

4.2.2 Rule Manager

The Rule Manager module is responsible for managing the rule data storage. The rules are stored in the JavaScript Object Notation (JSON) format, and each .json file is dedicated to a supported language (Java, Kotlin, XML). Rule Manager provides APIs to encode and decode JSON to Rule object. The *RuleManager* class implements the singleton design pattern to ensure that only one instance of it exists. This instance holds a list of rules for each language that are loaded from JSON files. The rules are cached in the list to avoid repeated file I/O and to save memory.

4.2.3 Rule Editor

ArmorDroid lets you adjust the rules to fit your requirements. It has UI features that can be found in the Tools tab of the toolbar, under the "ArmorDroid

Secure Coding" menu option. These UI features are written in Java using custom Swing UI toolkit components. Using these custom components ensures that our plugin's UI matches the IDE's UI in appearance and functionality.

4.3 Rule Model

Each security issue that we have collected is turned into a *Rule*. *Rule* is a data model that we have designed to contain information about a vulnerability and how the plugin interacts with that insecure code. The data structure of the *Rule* class is presented in Figure 5.

The *briefDescription* field of each *Rule* stores the message that appears when the user clicks on the code with a warning (see Figure 2). The *Rule* can be enabled or disabled by setting the boolean value *enabled* to *true* or *false*. The *inspector* field of type *Inspection* holds the regex patterns that check the security of a code fragment. It consists of a string *pattern* and a list of patterns for captured groups (gro, 2023). The main pattern is matched first. If it matches, the captured groups are matched with the corresponding pattern in *groupPatterns*. If any of the group patterns fail to match, the inspection result is *false*, indicating that the code is secure. Otherwise, if all the patterns match, the inspection result is *true*, and the code is flagged as insecure.

A *Rule* can provide none, one, or more than one solution to an issue. It has a field *fixes* that contains an array of *ReplaceStrategy* objects. Each object has a *fixName* field that shows the name of the fix on the

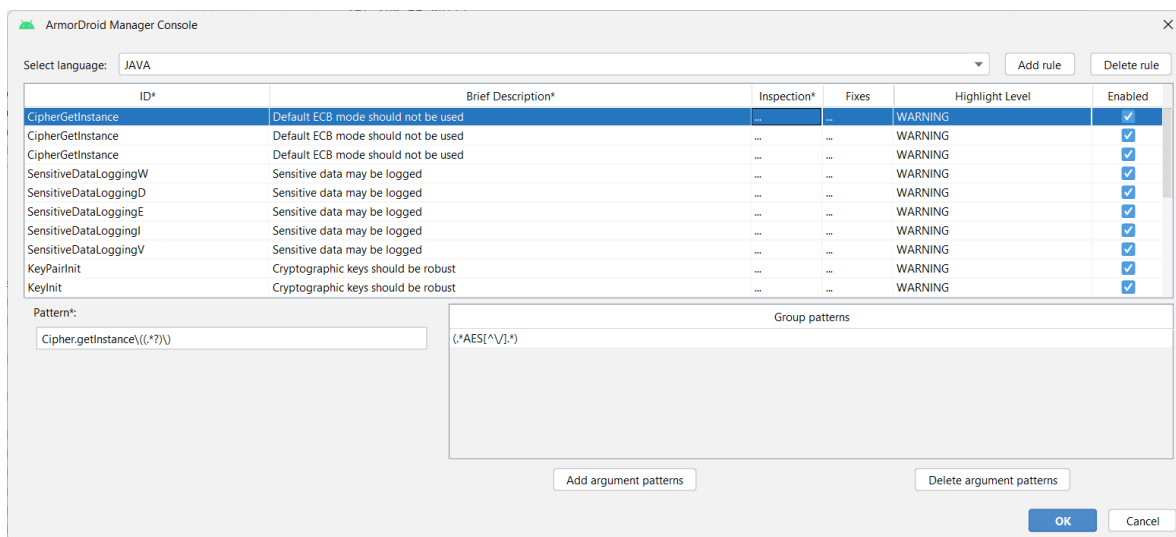


Figure 4: Rule customization dialog.

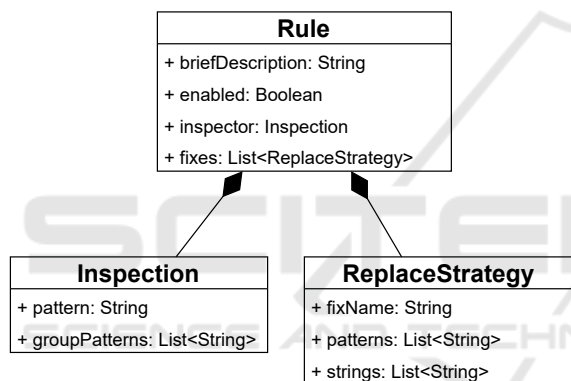


Figure 5: Data model of a rule.

suggestion panel (Figure 3). The fields *patterns* and *strings* are arrays of regex patterns and replacement strings, respectively. ArmorDroid uses them to replace insecure code with secure ones.

4.4 Rule Checklist

ArmorDroid default rule set contains 60 detection patterns covering 3 programming languages (25 for Java, 25 for Kotlin, and 10 for XML). 48 out of 60 rules support quick fixes, the ones without quick fixes are those that are too complicated to fix correctly or are not serious enough. A snippet of the issue list is shown in table 1.

Our collection of rules comes largely from our own knowledge and experience with insecure coding practices in Android app development. During our research, we also supplement the rule set with some rules from Sonarlint’s SAST rule list (son, 2023a).

4.5 Examples of Usage

4.5.1 Code Inspection and Quick Fix

ArmorDroid helps developers avoid insecure code by highlighting and suggesting fixes. For example, Figure 2 show a Cipher object with the AES/CBC/PKCS5Padding parameter, which is vulnerable to attacks. ArmorDroid marks this code with a warning and provides detailed information when the mouse is over it. The programmer can click on the light bulb icon to see a list of quick-fix options and choose one to replace the insecure code.

4.5.2 Rules Customization

ArmorDroid allows users to customize inspection rules for each projects; for example, users might want to detect the use of RSA without OAEP padding scheme. Users can do this by choosing the ArmorDroid Secure Coding → Customize Rules to open the rule editing dialog.

To create a custom rule, press the Add rule button. A new row will appear at the end of the table. Enter the details of the vulnerability you want to find and fix in this row. You can also edit or delete existing rules (Figure 4).

4.5.3 Rules Import/Export

ArmorDroid is a tool that allows developers to create and share their own rules for Android development. These rules can be saved as *.armor* files and loaded by other developers in their Android Studio. Alternatively, the rules can be distributed through version control systems as JSON files in the project.

Table 1: A part of ArmorDroid’s rules.

Issue	Language	Regex	Description	Quick fix
Driver Manager Get Connection	Java	<code>DriverManager.getConnection\((.*?)\,(.*?)\,(.*?)\)</code>	A secure password should be used when connecting to a database	
Password Encoder	Java	<code>StandardPasswordEncoder\((.*?)\)</code>	Passwords should not be stored in plain-text or with a fast hashing algorithm	Use BCryptPasswordEncoder instead of StandardPasswordEncoder
SSL Context	Java	<code>SSLContext.getInstance\((.*?)\)</code>	Weak SSL/TLS protocols should not be used	Improve security by using TLSv1.2
Message Digest	Java	<code>MessageDigest.getInstance\((.*?)\)</code>	SHA-1 and Message-Digest hash algorithms should not be used in secure contexts	Improve security by using SHA256
RSA Wrong Mode And Padding Scheme	Kotlin	<code>Cipher.getInstance\((.*?)\)</code>	Encryption mode and padding scheme should be chosen appropriately	Use OAEP padding scheme
Weak SSL/TLS Protocols Get Instance	Kotlin	<code>SSLContext.getInstance\((.*?)\)</code>	Weak SSL/TLS protocols should not be used	Enforce TLS 1.2 as the minimum protocol version
Accessing Android External Storage	Kotlin	<code>Context.getExternalFilesDir\((.*?)\)</code>	Accessing Android external storage is security-sensitive	Use internal storage by getFilesDir
Broadcasting Intents	Kotlin	<code>Context.sendBroadcast\((.*?)\)</code>	Broadcasting intents is security-sensitive	Restrict the access to broadcasted intents
Enabling File Access for WebViews	Kotlin	<code>WebView.getSettings().setAllowContentAccess\((.*?)\)</code>	Enabling file access for WebView is security-sensitive	Disable access to local files for WebViews unless it is necessary
Debug Features Activated	XML	<code>android:debuggable = \\\"true\\\"</code>	In the application manifest element of an android application, setting the debuggable property to true could introduce a security risk	In AndroidManifest.xml the android debuggable property should be set to false
Clear Text Protocols	XML	<code>android:usesCleartextTraffic = \\\"true\\\"</code>	Clear-text protocols (ftp, telnet, http) lack encryption of transported data, as well as the capability to build an authenticated connection	Disable all clear-text traffic

5 PILOT STUDY

5.1 Study Tasks

In our study, we invited Android developers ranging from a few months to a few years of experience to solve a coding exercise comprised of 5 sub-tasks with and without the assistance of ArmorDroid. The exercise involves an Android project that uses cryptographic APIs to encrypt and decrypt the input string. But the encrypting and decrypting code was missing and the programmers had to implement it. To evaluate ArmorDroid’s effectiveness, the participants received a guidance document on how to use the tool. After the exercise, the participants were asked to fill in a survey form that collected personal and technical data to evaluate the tool’s effectiveness.

5.2 Study Results

5.2.1 Participants

Our study involved 41 participants from various domains, including students and developers (see Table 2). Almost all of them had Android development experience of various levels: 24 participants have less than 1 year, 12 participants between 1 year and 2 years and 5 participants have more than 2 years.

Table 2: Participant’s Background.

Participant’s Background	Count
Student	20
Android Developer	9
Security Developer	2
Other	10

According to our survey, about 68% of the participants have no experience in security. However, roughly 76% of them are still concerned about security issues that stem from the coding process. This indicates that most developers prioritize secure coding but have not received adequate security training.

5.2.2 Features

Code Inspection. As shown in Figure 6, ArmorDroid code inspection feature was effective in detecting vulnerabilities code during the plugin development. The plugin highlighted the vulnerable code lines in the editor and provided a brief description of each error. This helped users to identify and understand the vulnerabilities code easily.

However, the brief note on the issues did not help developers gain insight into the issue. Out of 41 de-

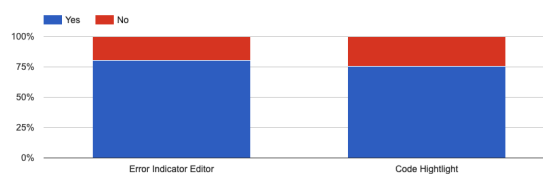


Figure 6: Participants’ Experience with Code Inspection.

velopers, less than 10 strongly agreed, and less than 10 agreed that the note was useful. About 17 were neutral. The brief note should include links to relevant documentation that provide a detailed view of the problem for developers.

Quick Fix. The quick fix feature was a popular option among the participants. According to the survey results, more than two-thirds (71%) of them indicated that they utilized this feature during the experiment. The results show that the majority of the participants have used quick fixes and achieved good results.

Rule Manager. About 90% of the participants reported that the Rule Manager was easy to use. However, a small percentage of participants still do not understand the rules and suggest improving the UI of Rule Manager. In another question, we ask them whether customizing the rule set was easy. More than half of the participants (56%) think that customizing the rule set is difficult. This implied that more work needs to be done on the Rule Manager UI for our future work.

5.2.3 Performance

We have conducted a performance comparison between ArmorDroid and Sonarlint on a Windows 11 laptop device with 16 GB of RAM and an Intel Core i5-8300H CPU. The first experiment focused on the speed of detection, while the second experiment focused on the memory consumption.

For the speed experiment, we run the test on 30 samples of bad code and measured the time taken by each plugin to flag them. The results showed that ArmorDroid was about as fast as Sonarlint, with a range of 120 to 400 ms versus 150 to 420 ms. This result proves that ArmorDroid meets our expectation of catching bad code in real-time.

For the memory experiment, we used a Python script to monitor the memory usage of Android Studio every 2 seconds for 2 minutes while doing a coding task similar to the one in the survey in Section 5. We repeated this 10 times and calculated the average memory usage for each condition. The results showed that Android Studio without any plugin used 1689.25 MB of memory, while with ArmorDroid it used 1824.24 MB and with Sonarlint it used 1892.61 MB. This means that ArmorDroid increased memory

usage by about 135.01 MB, while Sonarlint increased it by about 203.36 MB. ArmorDroid used less memory than Sonarlint initially, but this may be due to its smaller rule set and fewer features and languages. Sonarlint has more features than ArmorDroid, which could explain its higher memory consumption.

6 CONCLUSION AND FUTURE WORK

This article presents ArmorDroid, a plugin for Android Studio that helps developers avoid common security problems in Android programming. ArmorDroid detects insecure code patterns in real-time and suggests fixes. It also allows users to customize and share inspection rules with co-workers. We evaluated ArmorDroid with junior Android developers and found that they appreciated its ability to identify and correct vulnerable code quickly and easily. They also found the rule editing feature very useful.

The survey reveals some areas for improvement in current ArmorDroid. First, the rule editor's UI. They complained that it needed to be clarified or that the regex pattern fields were not validated. Another issue is the code inspector's inability to inspect variable function arguments due to ArmorDroid's limited understanding of the expression context.

In future work, we plan to redesign the UI of the rule editor. We will also create a website where all the rule detail explanations and examples will be presented. In addition, the link to each issue will be attached to the brief description, and users can follow the link to see the vulnerability in more detail. Finally, we are going to improve ArmorDroid's context awareness, *i.e.*, allowing it to inspect the value of a variable.

ACKNOWLEDGEMENTS

This research is funded by the University of Science, VNU-HCM, Vietnam under grant number CNTT 2023-05

REFERENCES

- (2023). groupvalues in kotlin. <https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.text/-match-result/group-values.html>.
- (2023). Mobsf. <https://mobsf.github.io/docs/#/>.
- (2023). Plugin configuration file. <https://plugins.jetbrains.com/docs/intellij/code-inspections.html#plugin-configuration-file>.
- (2023a). Sonar rules. <https://rules.sonarsource.com/>.
- (2023b). Sonarlint. <https://www.sonarsource.com/products/sonarlint/>.
- App-Ray (2023). App-Ray. App-Ray website.
- De Cremer, P., Desmet, N., Madou, M., and De Sutter, B. (2020). Sensei: Enforcing secure coding guidelines in the integrated development environment. *Software: Practice and Experience*, 50(9):1682–1718.
- Fan, W., Zhang, D., Chen, Y., Wu, F., and Liu, Y. (2020). Estidroid: Estimate api calls of android applications using static analysis technology. *IEEE Access*, 8:105384–105398.
- Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., and Rajarajan, M. (2014). Android security: a survey of issues, malware penetration, and defenses. *IEEE communications surveys & tutorials*, 17(2):998–1022.
- Gitlab (2022). The GitLab 2022 Global DevSecOps Survey.
- Google (2023). Android studio. <https://developer.android.com/studio>. Accessed: February 19, 2023.
- GuardSquare (2023). AppSweep: Mobile Application Security Testing. GuardSquare website.
- Nguyen, D. C., Wermke, D., Acar, Y., Backes, M., Weir, C., and Fahl, S. (2017). A stitch in time: Supporting android developers in writing secure code. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1065–1077.
- OWASP (2016). M5: Insufficient cryptography. <https://owasp.org/www-project-mobile-top-10/2016-risks/m5-insufficient-cryptography>.
- Statista (2023). Quarterly number of mobile app downloads worldwide from 1st quarter 2016 to 4th quarter 2022. Statista website. Accessed: February 19, 2023.
- Talukder, M. A. I., Shahriar, H., Qian, K., Rahman, M., Ahamed, S., Wu, F., and Agu, E. (2019). Droidpatrol: a static analysis plugin for secure mobile software development. In *2019 IEEE 43rd annual computer software and applications conference (COMPSAC)*, volume 1, pages 565–569. IEEE.
- Tran, A.-D., Nguyen, M.-Q., Phan, G.-H., and Tran, M.-T. (2021). Security issues in android application development and plug-in for android studio to support secure programming. In *Future Data and Security Engineering. Big Data, Security and Privacy, Smart City and Industry 4.0 Applications: 8th International Conference, FDSE 2021, Virtual Event, November 24–26, 2021, Proceedings 8*, pages 105–122. Springer.