



Security Issues in Android Application Development and Plug-in for Android Studio to Support Secure Programming

Anh-Duy Tran^{1,2(✉)}, Minh-Quan Nguyen^{1,2}, Gia-Hao Phan^{1,2},
and Minh-Triet Tran^{1,2}

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

{taduy,tmtriet}@fit.hcmus.edu.vn, {1712694,1712420}@student.hcmus.edu.vn

² Vietnam National University, Ho Chi Minh City, Vietnam

Abstract. In the context of modern Android application development, security issues and secure programming are considered unignorable aspects to ensure the safety of Android applications while still guarantee the development speed. The lack of attention to security factors in the software development process or the delay of traditional security assurance methods are the main causes of unsafe Android software. Those unsafe Android applications contain many vulnerabilities and a high risk of leaking user information, especially since Android applications are rapidly developed and published. Developers must adhere to a secure development process to counter Android application risks to avoid data leakage or access control flaws. Security has to be integrated throughout the application development process to secure the software development life cycle. This paper presents two main research contributions: summarizing common security issues in Android applications and developing a plug-in for Android Studio to support secure programming, 9Fix. The low-time-cost 9Fix plug-in can inspect your vulnerable code and instantly suggest an alternative secure code for developers in programming time that helps to improve the security and instruct the developers on how to write a secure code. Moreover, the developers can add their own security rules to 9Fix so 9Fix can adapt smoothly in a specific situation. We also demonstrate the effectiveness and the convenience based on the student feedback by experimenting with the 9Fix plugin.

Keywords: Android security · Secure coding · Android Studio plugin · DevSecOps

1 Introduction

Smart devices are increasingly widely used and play an important role in all aspects of life, from improving product quality, increasing human connection,

and supporting a more comfortable life. It is not difficult to see that *smartphones* still hold the leading position in the market and are indispensable devices for each person. According to the worldwide mobile OS market share report by Statista [17] (a professional statistics platform with over 1,000,000 statistics on over 80,000 topics from over 22,500 data sources), in January 2021, Android and iOS are the most popular operating systems for smartphones, accounting for 99.4% of smartphones worldwide, of which Android accounted for 71.93%. Besides, application development for mobile devices is an extremely fast-growing industry. However, the number of applications developed at high speed also leads to countless less secure applications. Vulnerable apps are making smartphones less secure. The main reasons for that situation are:

- The software development market is relatively open and unrestricted, so any software developer, including those who lack knowledge of secure programming and ignore security vulnerabilities, can publish their software to the market.
- There are no security standards for mobile software development yet.
- Pressure to quickly push products to market causes companies not to pay much attention to security.
- Developers do not have much training in secure programming or have not formed security awareness.

According to Verizon’s 2021 Mobile Security Index [1], 45% of organizations sacrificed mobile security in the past year. The main justifications cited for sacrificing security were expediency and convenience, with COVID-19 making a special guest appearance. Many mobile app development teams are urged to put the time to market ahead of security. As a result, their apps are vulnerable to cyber-attacks. While the speed of development is crucial, many apps that were released too soon have been attacked. For example, the famous game *Pokemon Go* has a cheat that can exploit root access on Android devices. Even iOS apps are vulnerable to mobile attacks. Operating system-level requirements give some developers a false sense of security. However, these are intended to safeguard users rather than app developers and publishers.

Security issues on mobile devices are always a concern, especially for the Android platform, which accounts for most of the market and is arguably more open than iOS. Apps running on Android are easier to reach the market. Putting an application on Google Play is much simpler than the iOS App Store because the cost is low, and especially the censorship process is also much faster: about one day for Google Play and weeks for the App Store. Google Play is a rapid software approval that makes this market more dynamic, leaving security vulnerabilities stemming from insecure programming. Therefore, to ensure the safety of users when using their apps, Android developers need a secure software development process.

Along with Agile, DevOps is one of the most used software development processes because of its performance and speed. **DevOps** is a combination of **Development** and **Operations**. DevOps is a method for establishing close cooperation between the development and operation phases of software. Today, DevOps

is the key to optimizing time and resources for better productivity. Based on DevOps, we have a more advanced approach, DevSecOps, with the addition of **Security**. DevSecOps automates security integration at every stage of the software development cycle, enabling secure software development at the speed of Agile and DevOps. Previously, security is applied at the end of the software development phase and checked by separate quality assurance or a penetration testing team. This fact leads to a situation where security vulnerabilities can be ignored to ensure development progress. However, with the support of DevSecOps, security is integrated and performs parallel to development and operations, ensuring that vulnerabilities are detected earlier, remedied faster, and cost less. To integrate **Sec** into **DevOps** requires a lot of steps, which can be mentioned as pre-commit hooks, software component analysis (SCA), static application security testing (SAST), dynamic application security testing (DAST), etc. throughout the DevOps pipeline. In particular, static application security testing helps find potential vulnerabilities in the source code without executing the code, contributing to the safety of the software.

From the above problems, it is essential to find out the vulnerabilities in Android applications. Most of these vulnerabilities stem from the process of programming and developing applications. Moreover, we can also prevent these vulnerabilities right in the programming process through secure coding. A tool to support programmers right in the coding process is necessary and convenient to help programmers see vulnerable code, unsafe protocols and can fix them instantly. In this paper, our main contributions are:

- Research security vulnerabilities in Android applications. From there, synthesize common Android application attack scenarios and prevention methods.
- Build a list and categorize common Android vulnerabilities.
- Research Android lint and propose a new architecture for its Detection and Reporting System.
- Build plugins for Android Studio, the 9Fix, to support secure programming, which integrates the proposed security protection category in Android.
- Conduct experiments to test the effectiveness of the 9Fix plugin.

The content of this paper is organized as follows. In Sect. 2, we briefly review the research on security issues in Android applications and the tools that help developers ensure the security of Android applications. In Sect. 3, we introduce the common security vulnerability in Android apps, which will be used to develop a security ruleset. From that ruleset, we then develop in Sect. 4 an Android Studio plug-in, 9Fix, which helps developers to inspect and fix the insecure code at programming time. The experiments of our 9Fix are discussed in Sect. 5. Finally, we give our conclusion for this work in Sect. 6.

2 Related Work

We divide related work into two small parts: discussing the research on security problems in Android application development and summarizing the methods or tools that help developers write secure code.

2.1 Security Issues in Android Applications

From the viewpoint of application development, we mostly focus on the security issues in the application layer. Those issues can be affected or triggered by the Android malware, wrongly implementing API or library, privilege escalation attacks, etc. Ma et al. [11] conducted a static analysis to extract control and data dependencies and build a classification model to determine the authentication bugs in Android applications. That system successfully identifies 691 SSL/TLS authentication bugs. Using a static code analysis tool, Poeplau et al. [14] explored dynamic code loading in Android applications. Many programs load additional code in an insecure manner, according to their findings.

In the context of mobile-based user-interface (UI) attacks, much research showed that clickjacking could easily deceive users into clicking on malicious controls. For example, the research of Aonzo et al. [3] demonstrated the end-to-end phishing attack requiring only a few user's clicks. The ClickShield developed by Possemato et al. [15] employed image analysis techniques to determine whether the user could be confused in the phishing attack.

Android applications are also vulnerable to various assaults due to the integration of web pages into mobile apps. For example, Bao et al. [7] investigated Cross-site Scripting vulnerability on Hybrid mobile apps makes it possible for attackers to bypass the access control policies of WebView and WebKit to run malicious codes into victim's WebView.

Bagheri et al. [6] performed an analysis of the permission protocol implemented in Android and proposed a formal model of the Android permission protocol to identify potential flaws. The study found that weaknesses in the Android permission system can have serious security ramifications, allowing an attacker to bypass permission checks in some circumstances completely.

2.2 Android Secure Coding Methodologies and Tools

From performing static code analysis, we can inspect the control-flow to identify the possible execution path and the data-flow to specify the possible predicted values of variables at the location of execution [10]. A tool named StubDroid can automatically generate the summarized models of the Android framework or libraries by using the flow analysis developed by Arzt et al. [4]. Another research by developed Arzt et al. [5] FlowDroid, an open-source tool that can detect the data leakage in the source code of Android applications by also using static code analysis. However, this tool cannot fully be used to detect security bugs in Android apps like SQL injection and intent leakage. Those tools normally find the vulnerable code after the program has been written.

Another example is ComDroid which is developed by Chin et al. [8]. By following the approach of building secure applications and systems from platform-level, API-level, and design-level solutions, ComDroid can detect and alert developers of application vulnerabilities in communication. However, the downside of ComDroid still needs a compiled code to scan threats and thus cannot help developers at coding time.

The idea of building a tool to support secure coding in Android development is not new. There are many tools and research that have been published and tackle one or more security problems in Android applications and help the developers to write safe code [9, 16]. Most of those research base on static code analysis, which tries to analyze the source code without running its program. Recently, Tabassum et al. did a study that compared the impact of secure programming tool support (ESIDE) versus teaching assistants. The findings revealed that ESIDE offered students greater information regarding security issues. Therefore, tools are built in the form of an IDE plugin that can help the developers to inspect the insecure code at the right coding time shows their effectiveness. The Application Security IDE (ASIDE) and the Eclipse IDE extension highlight the potential security issues, bugs, or vulnerabilities in the code and help fix them at the development phase. Eclipse IDE plugin, SonarLint [19] gives real-time feedback for Python, JavaScript, and Java programming languages. The Snyk [19] from Eclipse plugins can examine code dependencies using dependency trees, as well as check for vulnerabilities and recommend patches. However, those tools and plugins cannot support or integrate with Android Studio, one of the most used IDEs for Android development. FixDroid, which is developed by Nguyen et al. [13], is an Android Studio plugin that can help secure coding. It integrates many security tooltips to overcome some security weaknesses in the code at the programming time. Nevertheless, the developers cannot extend the rules or add more tooltips for new or special security issues. Moreover, the number of security rules in FixDroid is small, and it needs a server to handle the security problem sent by the client machine. That makes the privacy of the coding is not ensure.

To build an Android Studio plugin, this paper aims to employ a larger security tooltip set gathered from the common Android application security issues. This plugin can run locally in the developer's machines to remove the need for a handling server and thus guarantees the secrecy and privacy of the application's source code. Moreover, the developers can also extend security rules for this plugin.

3 Security Issues in Android Applications

In this section, we briefly summarize the common vulnerability in Android applications. From there, we can base on those security errors to build the tool that supports secure coding for Android Studio.

3.1 Security in Cryptography Implementation

Cryptography ensures the secrecy of the sensitive data which the application needs to store, analyze or transmit. The wrong way of implementation could make cryptography algorithms weak and vulnerable. Therefore, the developers have to follow the standard guidelines of implementation for each algorithm, and many requirements must be held to guarantee the safeness and performance of cryptography. First, developers should use strong cryptography algorithms

instead of outdated or weak ones. For example, use AES replace for DES, RC2, RC4, etc. Second, effective operation modes and padding schemes should be applied with the corresponding algorithms (GCM for AES, OAEP for RSE, etc.). Third, the length of the key should qualify the standard such as over 128 bit for block cipher and 2048 for RSA.

3.2 Security in Client-Side and Click-Jacking Prevention

The first security to be considered on the client-side is control database queries to avoid SQL Injection vulnerabilities. When querying to the database, developers often concatenate input strings from the user. These queries are likely to have SQL Injection errors. An attacker can view, add, modify, delete data, steal user information. To fix this, the developers have to use **PreparedStatement** so that parameters must be added with a function (setParam) to eliminate string concatenation. In the code below, username and password are parameterized when entering the query to avoid SQL Injection vulnerabilities.

```

1  SQLiteDatabase db = mDbHelper.getWritableDatabase();
2  String userQuery = "SELECT * FROM useraccounts WHERE user_name
   = ? and password = ?";
3  preparedStatement.bindString(1, Username.getText().toString() ("
   user_name");
4  preparedStatement.bindString(2, Password.getText().toString() ("
   password");
5  SQLiteStatement preparedStatement = db.compileStatment(userQuery)

```

Second, the Android application should prevent XSS vulnerabilities. WebView is a key component in both Android and iOS platforms, allowing smart-phone apps to embed a simple yet powerful browser within them. By default, WebView disallows javascript execution, but if the programmer sets **setJavaScriptEnable(true)** property then WebView will allow javascript execution and can exploit XSS error. When using WebView, it is not recommended setting property **setJavaScriptEnable (true)**.

Third, the Android application should also prevent Click-Jacking vulnerabilities. An attacker may use interface spoofing to trick the user into performing something hidden underneath the interface. When defining important layout elements, it is necessary to set the **filterTouchesWhenObscured** property to prevent that vulnerability.

3.3 Security in Communication Between Application and Server

Most Android applications have connections to servers located on the Internet to update and exchange information. This transmission may be eavesdropped on from telecommunications networks, Wifi networks, etc. Sensitive information is transmitted between the application and the server, such as login process, cookies, sessions, etc., must be encrypted.

TLS (Transport Layer Security) and SSL (Secure Sockets Layer) are standard technologies for keeping Internet connections secure and protecting data transferred between two systems, preventing hackers from reading and modifying any data being transmitted. A certificate containing the public key from the server is needed to enable SSL/TLS. In addition, the client must validate the certificate to ensure that the certificate is sent from the correct connected server. The Android operating system provides a built-in digital certificate authentication method, and developers can also build their certificate authentication method depending on their use. However, Android developers may not properly implement SSL/TLS during application development, leading to Man-in-the-middle attacks or user phishing attacks [20].

In Android development, we can create an HTTPS connection by using **HttpsURLConnection**:

```

1  URL url = new URL("https://bank.com/login");
2  HttpsURLConnection urlConn = (HttpsURLConnection)url.
    openConnection();
3
4  //Or
5  HttpsURLConnection urlConn = new HttpsURLConnection("https
    ://bank.com/login");

```

3.4 Security in Android Components

Android components are the essential foundation of an Android application. Each component is an entry point through which the system or user can interact with the application. Some components depend on others. Android has four different application components: Activity, Service, Broadcast Receiver, and Content Provider.

When developing applications that allow other applications to interact with Activity, if the programmer uses the **android:exported= “true”** attribute or defines the **intent-filter**, without setting permissions to call this Activity, the attacker can take advantage of building malicious applications and call this Activity, then perform destructive, unwanted actions. For example, when an app needs to call another app’s Activity to perform a certain action, Android will allow the user to select apps with the same Activity without specifying which program’s Activity. Then the attacker can take advantage of an Activity with the same name to deceive users and steal information. Therefore, developers should set the permission for an Activity, so when you want to call that Activity, you should ask the permission.

Service is the entry point for the general purpose of keeping an application running in the background. It is a component that runs in the background to perform long-running operations or work for remote processes. Normally, the Service often handles sensitive information not related to the user interface, such as authentication with username, password, information synchronization, etc. If mistakenly called to Malware’s Service, sensitive information will be leaked. To

prevent this risk, developers should add permission when defining the Service and check the Service's permissions before passing sensitive information.

Content Provider allows the definition and access to shared data. However, if the permission check is not well controlled, the user can access data without authorization. When sharing data with other applications, minimum access control is required to avoid unauthorized access to sensitive data.

Broadcast Receivers allow the system to deliver events to the application outside of the normal event stream, allowing the application to respond to broadcast messages system-wide. Broadcast Receiver can work even when the application is inactive. Suppose the developer does not set up proper permissions when subscribing to the broadcast. In that case, the attacker can use the Broadcast Receiver to send spoofed events to deceive the user like fake SMS, or the attacker can steal the sensitive information sent via Broadcast Receiver. To prevent this risk, as a precaution, in case it is not necessary, developers must not set the **android:exported= "true"** attribute for the Broadcast Receiver and have to set the permission for receiving and sending Broadcast Receiver.

3.5 Protect Data Stored on Mobile Phones

When developers want to store data in Android phones, they should ensure the security of that data. The developers should decentralize and encrypt data stored on the device. The data stored temporarily in the phone in a file or SQL lite can be exposed if the phone is lost, the device is rooted, etc. Some rules should hold such as: restrict storing information on the phone, only save broadcast information; do not store sensitive data such as accounts, passwords, PINs, etc. in plaintext; you can use Android's Keystore to increase security; storing confidential company data and documents on mobile devices is strictly prohibited; do not use world readable and writable permissions when saving important information.

The developers also should protect sensitive data when apps are running in the background. When an application goes into the background, there may be a lot of information displayed on the interface that could be exposed if the user does not completely turn off the application and can be accidentally seen by someone. For example, when logging in with credit card information. When it is necessary to ensure that data is not batched after switching to the background, it is necessary to force the application event to go into background mode to hide this information.

3.6 Secure Coding with Logs and Debug Information

When programming or debugging applications, programmers often log or put some debug information that could reveal sensitive information. With logging logs as below, it will log both the login information and the user's password when the login fails. Doing so will reveal the user's username and password. To avoid this, log information is not recorded with important user information (password, etc.).


```
1  try {  
2  ...  
3  } catch (Exception e) {  
4      Log.w("Failed login", userName + " " + password);  
5  }
```

Besides, developers should turn off debugger mode when compiling the source code to strip debug symbols that may contain valuable information for attackers in reverse engineering. To do this, developers have to set the attribute `<application android:debuggable="false"/>` in the `Android-Manifest.xml` file.

4 Secure Coding Plugin for Android Studio

Android Studio [2] is an IDE that was announced in 2013 at the Google I/O conference, built on the IntelliJ Platform by Google. It is professionally designed to develop apps for the Android operating system. It provides all the principal functionality that an IDE must-have for supporting app development. Android Studio has an editor that can do syntax checking and code completion, and the real power comes from the Program Structure Interface (PSI). PSI provides a lot of functionality, from quick navigation to files, symbols to the quick fix, refactor the source code, and many other functions.

4.1 Android Lint

In the past, programmers had to spend a lot of time detecting a simple error such as using an uninitialized variable or index out of range. Therefore, after the C language was founded, creating a tool to help detect potentially error-prone code and reduce the burden of reading code for programmers. In 1987, Stephen C. Johnson [12] - a computer scientist at Bell Labs created lint to help with debugging of one of its projects, and then lint had been widely used outside of the lab.

The lint tool with its amazing capabilities is growing and becoming popular to this day. Vendors also create many different lint variants for their projects, including Google with Android lint. It is a tool specifically for Android source code analysis that supports software developers to analyze all components in a project, including source code, configuration files, resource files, and helps developers to automatically detect and edit issues in the project without having to execute the application.

Android lint can be used manually by a command line like the original lint tool or a standalone lint tool. Project's source code and the configuration file **lint.xml** that will be loaded into a scan engine and produce *Issues*. Besides, Android lint in Android Studio is usually used as an *Inspection* that allows developers to control rules thanks to the interface panel. Especially, Inspection can also suggest developers edit problematic source code when programming that other ways cannot do. This capability is very powerful for our team to

build a system that real-time detects errors and suggests modifications to the source code.

4.2 Proposed Architecture for Extending Android Lint’s Detection and Reporting System

Lint scans Android source code for Issues. There are many types of Issues, which link with metadata like id, category, brief explanation, severity, and so on. To be enabled, the Issue must be registered with lint’s issues management system called the Issue Registry. An Issue is an object whose main function is to contain information about an Issue, so each issue needs a partner to interact with source code and users. This partner is *Detector* that can be responsible for finding problems and reporting them to lint.

A Detector can implement a scanner for a specific scope. There are many scanning toolkits such as ClassScanner, GradleScanner, BinaryFileScanner, etc. Still, to detect security vulnerabilities in the programming process, we only focus on XmlScanner to analyze XML resource files and UastScanner to analyze Java source code. When a Detector reports a problem, it can provide a *Lint Fix*. This Lint Fix class has the main task of finding and replacing an old code with a new code in the same location. The code that needs to be replaced can be searched using a regular expression, making editing flexible and retaining part of the context of the source code. The relation between concepts is modeled as shown in Fig. 1.

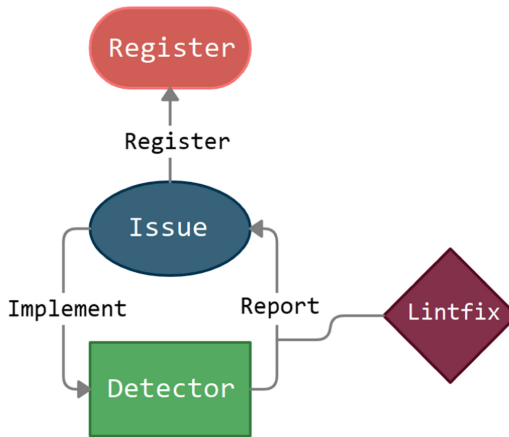


Fig. 1. The relation between lint’s concepts

In general, the lint’s rules of detecting and reporting working are shown in the Fig. 2. The system consists of many Issues. Each Issue points to a Detector object class that detects problems and report for that Issue which leads to

every time developers want to add or edit custom rules to inspect code for their projects, they must implement a couple of issues, detector, and register to the Issue Register.

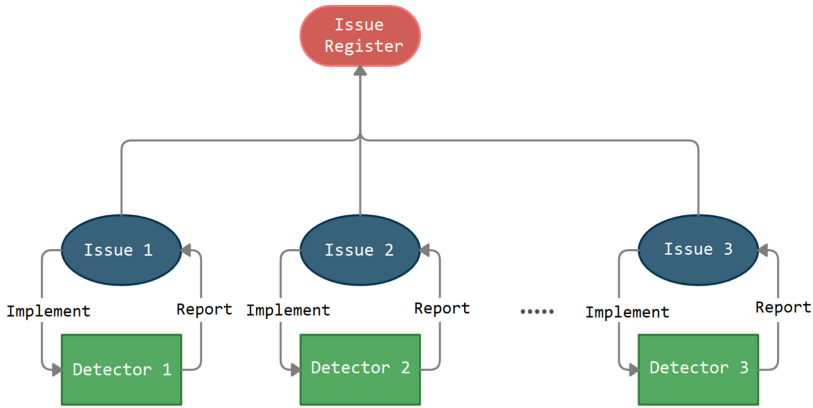


Fig. 2. Android lint's detection and reporting system

From the stated limitation, our team decided to find a way to generalize and group the rules based on the detection. We can save all the Issues' information, the Detectors' problem code and Lint Fix's pattern to generate a quick fix for the developer of each problem rule to the storage. Our new system is described in the Fig. 3.

The Issue Entry components are responsible for declaring the Detector groups. Each Detector will scan and check an entire group of problems divided based on the detection method. Developers can edit or add their rules in the Rules Storage. After that, Detector groups will load detection rules, report Issues if they meet the condition and craft Lint Fix. To provide a basic set of safe programming rules and allow developers to customize each project, our team decided to build a plugin for the most popular Android app development tool, Android Studio.

4.3 9Fix: Android Studio Secure Coding Plug-in

Android Studio, designed base on The IntelliJ Platform, has extremely powerful extensibility. There are several types of plug-in, but two of them are suitable for the tool our team is planning to develop. **UI themes** provides the ability to edit some component protocols IDE interface such as MenuBar, ToolBar, which can provide developers dialogues for interacting with plugins. **Custom language support** syntax highlighting, language testing, parsing source code, and more. Gathering all the techniques and proposed architecture, we composed a secure coding plug-in for Android Studio, the **9Fix**. Our plug-in architecture is described in Fig. 4.

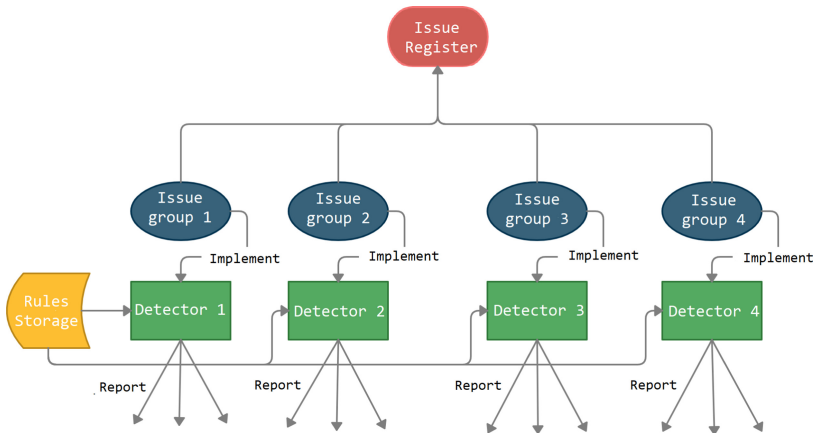


Fig. 3. Android lint's detection and reporting system with rules storage

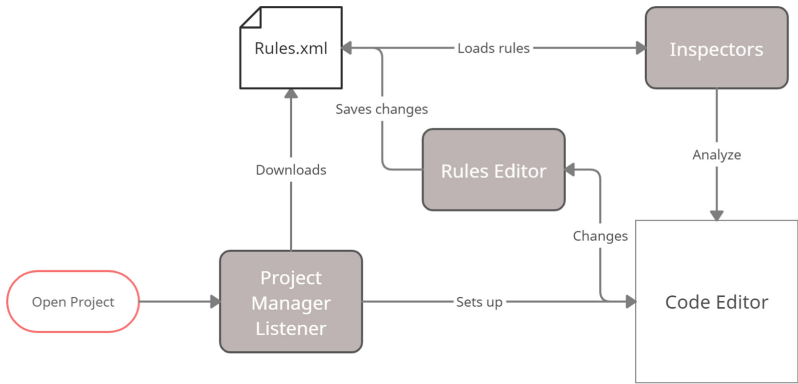


Fig. 4. Architecture of 9Fix

The Project Manager Listener always listens to the opening project event, then automatically downloads the default rules storage file called Rules.xml that contains problems, detection phrases, and recommend code pieces. This file and our Inspection module will be stored and work locally, do not send any data to any web sever that keeps the project source code completely secret. Actions and Dialogues are responsible for interacting with the user and changing the ruleset so that the Inspection can load changes and immediately affect the code editor.

4.4 Example of Use

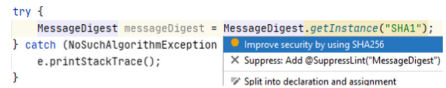
Figure 6 and 5 show how 9Fix detected the problem and helped the developers fix it. When developers create MessageDigest with the SHA-1 parameter, 9Fix will highlight the code for users' attention. The Issue will be displayed in detail every time the programmer moves the mouse and suggests the user fix the error using

the safer parameter “SHA-256”. However, programmers do not always notice, or they clone a big project with many files. They can use the Inspect Code feature to scan the entire source code. The pieces of code that our team considers to be problematic will also be detected along with the available rules of Android Studio. We tested with Tencent’s open source project VasSonic [18] and scanned the unsafe code listed in Fig. 7.



```
try {
    MessageDigest messageDigest = MessageDigest.getInstance("SHA1");
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
}
```

Fig. 5. 9Fix suggests fixing for programmers

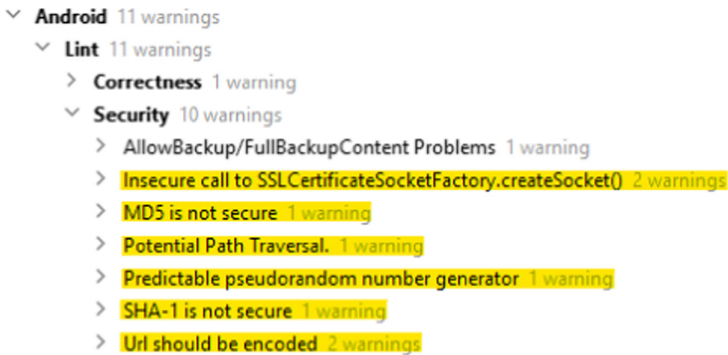


```
try {
    MessageDigest messageDigest = MessageDigest.getInstance("SHA1");
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
}
```

Fig. 6. 9Fix detects problem

4.5 Rules Customization for a Specific Project

9Fix’s architecture allows Inspection to handle all work locally and follow all information in the Rules.xml file stored within each project. Users have full editing rights through the dialog box; for example, users who want to detect more MD5 parameters and edit suggestions to SHA-256 can add their own rules. Immediately when the content of Rules.xml changes, Inspection will update the ruleset and detect the code that uses the MD5 parameter, demonstrated in Fig. 8.



```

Android 11 warnings
  Lint 11 warnings
    Correctness 1 warning
    Security 10 warnings
      AllowBackup/FullBackupContent Problems 1 warning
      Insecure call to SSLCertificateSocketFactory.createSocket() 2 warnings
      MD5 is not secure 1 warning
      Potential Path Traversal. 1 warning
      Predictable pseudorandom number generator 1 warning
      SHA-1 is not secure 1 warning
      Url should be encoded 2 warnings
  
```

Fig. 7. 9Fix scans the security errors of entire VasSonic’s source code

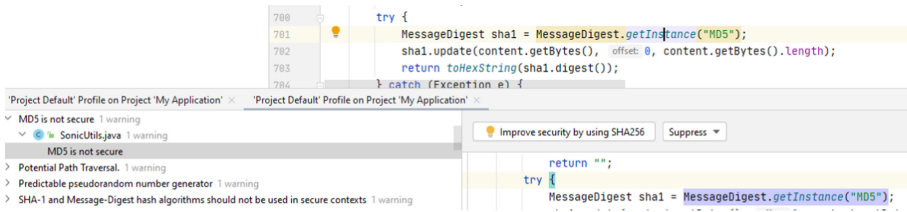


Fig. 8. Customized rules to detect MD5

4.6 Security Checklist

9Fix currently contains more than 50 detection patterns and supports quick fixes to cover 25 popular security Issues. Some Issues help users fix errors, and others remind programmers to use filters or be careful when using them directly. The list of security Issues and Warnings is shown in the Table 1.

Table 1. Common security issues and warnings list

Issue	Problem object	Explanation	Quick fix
CipherGetInstance	Cipher	AES and GCM mode should be used to improve security	Improve security by using AES/GCM/NoPadding
KeyPairInit	KeyPair-Generator	Key length should be 2048	Improve security with 2048-bit key
KeyInit	KeyGenerator	Key length should be 128	Improve security with 128-bit key
ECGenParameter-Spec	ECGen-ParameterSpec	Cryptographic keys should be robust	Improve security with secp224k1
DriverManager-Connect	DriverManager	A secure password should be used when connecting to a database	
PasswordEncoder	Standard-Password-Encoder	Passwords should not be stored in plaintext or with a fast hashing algorithm	Improve security with BCryptPasswordEncoder
SSLContext	SSLContext	Weak SSL/TLS protocols should not be used	Improve security by using TLSv1.2
NullCipher	NullCipher	NullCipher should not be used	
MessageDigest	MessageDigest	SHA-1 and Message-Digest hash algorithms should not be used in secure contexts	Improve security by using SHA256
PredictableRandom	Random	The use of a predictable random value can lead to vulnerabilities	Use SecureRandom
PathTraversal-Potential	File	A path traversal attack (also known as directory traversal) aims to access files and directories that are stored outside the root folder	Add path traversal filter
CommandInjection	Runtime	Execute command with user input may trigger command injection	

(continued)

Table 1. (*continued*)

Issue	Problem object	Explanation	Quick fix
UntrustedServlet	HttpServlet-Request	You may need to validate or sanitize anything pulled from it before passing it to sensitive APIs	
AndroidExported	exported	The Android application exports a component for use by other applications, but does not properly restrict which applications can launch the component or access the data it contains	Change to <code>android:exported="false"</code>
MissingUrlEncode	URL	Proper input sanitization can prevent the insertion of malicious data into a subsystem such as a database	
ScriptCodeInjection	ScriptEngine	Eval method with user input may trigger command injection	
XPathInjection	XPath	The parameter in compile method should be whitelisted	
PatternLoad	Pattern	Remember to use double slashes for escape characters	
FileUpload	FileUtils	Metadata should be checked to prevent unsafe file upload	
SensitiveCookie	HttpCookie	Do not store unencrypted sensitive information on the client side	
ReadUnshared	ObjectInput-Stream	<code>readUnshared()</code> may produce unexpected results when used for the round-trip serialization	Use <code>readObject()</code> to ensure that the object referred to only one target
WriteUnshared	ObjectOutput-Stream	<code>writeUnshared()</code> may produce unexpected results when used for the round-trip serialization	Use <code>writeObject()</code> to ensure that the object referred to only one target
LogPassword	Log	The application is deployed with debugging code still enabled, which can create unintended entry points or expose sensitive information	
DebugEnabled	Debuggable	Information written to log files can be of a sensitive nature	Turn off debugger mode

5 Experiment Results

5.1 Performance

9Fix checks source code in real-time, so our team measures how quickly problem code is marked up after developers create them. The experiment was conducted on an 8 GB RAM device with a 2.30 GHz processor. The results of 100 measurements show that 9Fix can detect in the period from 100 to 500 ms. The results are similar when the user proceeds to select a quick fix or customize the rules.

5.2 User Experience

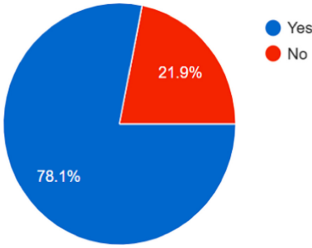


Fig. 9. Number of users who received 9Fix's suggestions

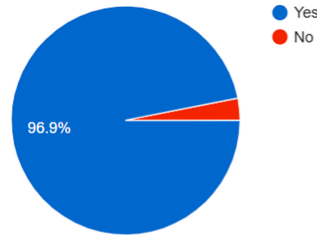


Fig. 10. Number of users interested in editing the rule set

We surveyed 30 Android developers with about 1 to 2 years of experience and 9Fix features. More than 76% of users claim to receive 9Fix warnings or problematic source code hints in a short period. A unique feature of 9Fix is to allow users to customize their own rules that are well-reviewed by users. Most of them show interest in the ability to customize rules for their projects (Figs. 9 and 10).

6 Conclusion

This scientific article summarized the common security problems in Android programming and developed a tool called 9Fix, based on Android lint, which helps detect insecure code immediately during the programming phase. Users can 9Fix to scan the vulnerabilities in the entire source code as well. We also proposed and implemented the new architecture for Android lint, so developers can add their security rules to 9Fix for extending the ruleset suitable for their project. Moreover, 9Fix also ensures the privacy of developers thanks to its serverless properties. We collected feedback from junior Android developers to show the effectiveness of our Android Studio's plugin. To be concluded, Android secure coding and the IDE plugin for secure programming are vital to guarantee security in Android development.

Acknowledgements. This research is supported by research funding from Faculty of Information Technology, University of Science, Vietnam National University - Ho Chi Minh City.

References

1. Mobile security index. <https://www.verizon.com/business/resources/reports/mobile-security-index/>
2. Android studio and sdk tools: Android developers. <https://developer.android.com/studio>

3. Aonzo, S., Merlo, A., Tavella, G., Fratantonio, Y.: Phishing attacks on modern android. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 1788–1801 (2018)
4. Arzt, S., Bodden, E.: Stubbroid: automatic inference of precise data-flow summaries for the android framework. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 725–735. IEEE (2016)
5. Arzt, S., et al.: Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM Sigplan Not.* **49**(6), 259–269 (2014)
6. Bagheri, H., Kang, E., Malek, S., Jackson, D.: A formal approach for detection of security flaws in the android permission system. *Formal Aspects Comput.* **30**(5), 525–544 (2017). <https://doi.org/10.1007/s00165-017-0445-z>
7. Bao, W., Yao, W., Zong, M., Wang, D.: Cross-site scripting attacks on android hybrid applications. In: Proceedings of the 2017 International Conference on Cryptography, Security and Privacy, pp. 56–61 (2017)
8. Chin, E., Felt, A.P., Greenwood, K., Wagner, D.: Analyzing inter-application communication in android. In: Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, pp. 239–252 (2011)
9. De Cremer, P., Desmet, N., Madou, M., De Sutter, B.: Sensei: enforcing secure coding guidelines in the integrated development environment. *Softw. Pract. Exp* **50**(9), 1682–1718 (2020)
10. Fan, W., Zhang, D., Chen, Y., Wu, F., Liu, Y.: Estidroid: estimate API calls of android applications using static analysis technology. *IEEE Access* **8**, 105384–105398 (2020)
11. Ma, S., Liu, Y., Nepal, S.: Are android apps being protected well against attacks? *IEEE Wirel. Commun* **27**(3), 66–71 (2020)
12. Morris, R.: Stephen curtis johnson: Geek of the week (2016). <https://www.red-gate.com/simple-talk/opinion/geek-of-the-week/stephen-curtis-johnson-geek-of-the-week>
13. Nguyen, D.C., Wermke, D., Acar, Y., Backes, M., Weir, C., Fahl, S.: A stitch in time: Supporting android developers in writing secure code. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1065–1077 (2017)
14. Poeplau, S., Fratantonio, Y., Bianchi, A., Kruegel, C., Vigna, G.: Execute this! analyzing unsafe and malicious dynamic code loading in android applications. In: NDSS, vol. 14, pp. 23–26 (2014)
15. Possemato, A., Lanzi, A., Chung, S.P.H., Lee, W., Fratantonio, Y.: Clickshield: are you hiding something? towards eradicating clickjacking on android. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 1120–1136 (2018)
16. Riad, A.K., et al.: Plugin-based tool for teaching secure mobile application development. *Inf. Syst. Educ. J.* **19**(2), 2 (2021)
17. O’Dea, P.S.J.: Mobile OS market share 2021 (2021). <https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/>
18. Tencent: Tencent/vassonic: Vassonic is a lightweight and high-performance hybrid framework developed by tencent vas team, which is intended to speed up the first screen of websites working on android and IOS platform. <https://github.com/Tencent/VasSonic>

19. Vermeer, B.: 10 eclipse plugins you shouldn't code without (2021). <https://snyk.io/blog/10-eclipse-plugins-you-shouldnt-code-without/>
20. Wang, Y., et al.: Identifying vulnerabilities of SSL/TLS certificate verification in android apps with static and dynamic analysis. *J. Syst. Soft.* **167**, 110609 (2020)