Applied Information Technology And Computer Science Vol. 5 No. 1 (2024) 1330-1340
© Universiti Tun Hussein Onn Malaysia Publisher's Office

AITCS

Homepage: http://publisher.uthm.edu.my/periodicals/index.php/aitcs
e-ISSN :2773-5141

Mobile Application Clone Detection Using Digital Signature

Emir Fikri Roslan[1], Kamarularifin Abd Jalil[1*]

[1]Kolej Pengajian Pengkomputeran, Informatik dan Media,
Universiti Teknologi MARA, Shah Alam, Selangor, 40450, MALAYSIA

DOI: https://doi.org/10.30880/aitcs.2024.05.01.075
Received 27 June 2023; Accepted 15 August 2024; Available online 30 August 2024

Abstract*: With the increased usage of mobile devices around the globe, the development of mobile apps has also increased. Application installs have increased proportion to the number of mobile devices usage. But with the increase of mobile applications installed on every user's phone, not all are aware that they may be a victim of cybercrime. Some of the application has been cloned using reverse engineering and injected with custom codes with malware and virus. In this paper a method to detect mobile application clone using digital signature is proposed. This method works by sending data to the server to check for the original application signature and install application signature to verify if the application is safe to use. Hence the method of checking digital signature using Java method and using Java Native Interface that will call C/C++ library is implemented in order to avoid APK Signature Tampering.*

Keywords*: Mobile App, Clone, Digital Signature*

1. Introduction

The advent of cellular network or mobile network in the 21st century has also resulted in the production of smart devices such as smart phones which are multifunctional. These devices are used not only for communication but also helps people to earn, learn and have fun. All of this are possible with the help of mobile application or simply known as mobile app. Starting with simple game application such as "snake", mobile app has evolved into more complicated applications ranging from managing health, business, life to name few.

Unfortunately, the advantages of mobile apps have been exploited by adversaries by producing clones for these apps. App cloning is a method where the adversaries will produce the clones of the application which has malware for personal profit [1][2]. App cloning is a severe problem that does not only affect the developers, but also destroy the health of the Android app market, hence proving the importance of detecting app clones [3]. In many cases, users become victims of app clones when they

**Corresponding author: author@organization.edu.co*
This is an open access article under the CC BY-NC-SA 4.0 license.

download apps outside the play store due to the fact that such apps are free. However, users forget that apps like this usually have hidden malware developed by adversaries.

Before any software can be loaded on Google Play Store, it must be digitally certified with a certificate. This certificate is used by Android to identify the author of the application, and it does not have to be signed by a certificate authority. Self-signed certificates are frequently used in Android applications. The private key of the certificate is held by the app developer.

The main reason for safeguarding the Android app is to employ a produced certificate and digital "key" to create a unique, encrypted, and reasonably un-hackable signature. This demonstrates that the app came from the main developer and not from some other untrustworthy source.

2. Related Work

2.1 Reverse Engineering and Tampering

The open nature of Android is frequently praised as enabling entrepreneurs to create ground-breaking technologies. Only a few, however, are aware of the negative side of this openness that is vulnerability. However, Android provides app developers with a significant edge that most mobile operating systems do not. It is very straightforward for an app developer or reverse engineer to analyze the source code of an open-source framework since it is open source [4]. The source code is accessible at the Android Open-Source Project (AOSP) and anyone can modify the code. In Android, reverse engineering is the process by which most reverse engineers re-obtain source code with the goal of replicating the program, building something comparable to it, or identifying an app's flaws or improving its security.

Because the code in Android apps is not converted into machine code, it is constantly open to extraction and reverse engineering [5]. The vulnerable code may then be exploited for a number of purposes, which can be frightening for any professional mobile app company, including: Repurposing the code for personal gain Find flaws in the code, look for sensitive data that has been hardcoded in the code, add malware fishing and making changes to the existing applications.

Malware such as spyware, trojans, adware, and other forms of malware pose a major danger to Android apps. Although some malwares are not intended to do harm, others might have unintended and unwanted consequences such as localized denial of service, anomalous battery drain, and so on. Spywares, for example, can get access to a smartphone's camera or microphone module and relay data back to the attackers. Adware is a form of malware that sends damaging advertising to a specific group of individuals via current communication channels such as email, instant messaging, MMS, Bluetooth, or SMS.

Tampering is referring to the process of modifying a mobile app source code with malicious intent and repackaging it to look like the original app. The modified app is then posted onto third-party app stores and will be waiting for its prey to download it without realizing the risks [6].

The process of changing a compiled app, e.g., altering the code in binary executables, resources or Java bytecode, is known as patching [7]. There are many ways where patches can be applied. This includes by editing binary files in a hex editor and decompiling, editing, and re-assembling an app.

2.2 Code Injection

Code injection is a technique used to exploit a vulnerability in a software application or system by injecting malicious code into the target system [8]. This is done by tricking the system into executing the injected code, which can then be used to carry out unauthorized actions such as stealing sensitive data, modifying system settings, or taking control of the entire system. Code injection can be performed

using various methods such as SQL injection, cross-site scripting (XSS), buffer overflow attacks, and more.

The impact of a successful code injection attack can be severe, as it can compromise the security and integrity of the entire system, and can potentially allow an attacker to gain complete control over the targeted system. It is important for software developers to be aware of the risks associated with code injection and take steps to prevent it from occurring, such as implementing input validation and sanitization techniques, using secure coding practices, and keeping software systems up-to-date with the latest security patches and updates [9].

2.3 Digital Signature in Mobile Applications

A digital signature in mobile applications is a cryptographic technique that is used to verify the authenticity and integrity of digital messages, documents, or software applications. Digital signatures use public-key cryptography to ensure that the message or application being signed has not been tampered with and that it was indeed created by the sender [10].

In the context of mobile applications, digital signatures are used to ensure that the app has not been modified or tampered with during the distribution process. This is particularly important for mobile app stores, where third-party developers can submit their apps for distribution. By requiring that apps be signed with a digital signature, the app store can verify that the app was created by the developer and has not been modified or tampered with by a third party.

To create a digital signature, the app developer uses a private key to sign the app, which generates a unique digital signature. When the user downloads the app, the app store checks the signature to ensure that it matches the original signature created by the developer. If the signature is valid, the app can be installed with confidence that it has not been tampered with or modified.

Developers will use application signing to identify the app's creator and update it without having to create complex interfaces or permissions. The developer must sign every program that runs on the Android platform. Apps that attempt to install without being signed will be refused by Google Play or the Android device's package installer.

On Google Play, application signing connects Google's trust in the developer with the developer's faith in their program. Developers are aware that their program is sent to the Android device unaltered, and they might be held liable for its behavior.

On Android, signing an application is the first step towards putting it in the Application Sandbox. Various programs run under different user IDs, and the signed application certificate indicates which user ID is connected with which application. Application signing guarantees that only well-defined IPC allows one application to communicate with another.

The Package Manager validates that the application (APK file) has been correctly signed with the certificate provided in that APK when it is installed on an Android device. If the certificate (or, more precisely, the certificate's public key) matches the key used to sign any other APK on the device, the new APK can declare in the manifest that it will share a UID with other similarly-signed APKs (refer to Figure 1).

Applications can be signed by a third party (OEM, operator, alternative market) or by the applicant themselves. Android supports code signing through the use of self-signed certificates, which developers may produce without the need for external assistance or authorization. A central authority is not required to sign applications. Currently, Android does not do CA verification on application certificates.

Because the new format is backwards compatible, APKs signed using the new signature format may be loaded on earlier Android devices (which simply disregard the extra data added to the APK), as long as these APKs are also signed with the v1 format.

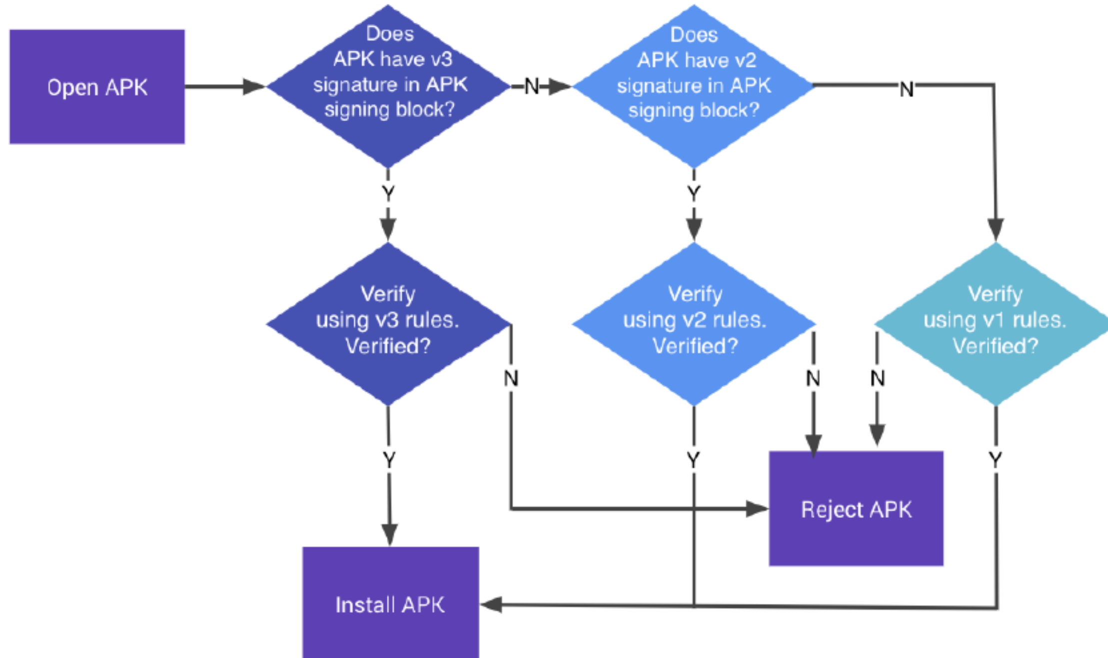

Figure 1: APK signature verification process

3. Methodology

In this paper, the standard software development process was used. There are six phases which are Information Gathering Phase, Planning & Design Phase, Implementation Phase, Testing Phase, Result & Analysis Phase and Documentation.

3.1 Information Gathering Phase

In the information gathering phase, much related information is collected and analyzed from articles, books, journals and on the World Wide Web. The information gathering is the most important method because it shows what would be the method used on the project and to ensure projects are on the right path to success.

3.2 Planning and Design

In this phase, the related software and environment were identified. This is followed by implementation of Research based on environment. Next, the amount of scenario to be created was determined. Lastly, select a proper solution if a vulnerability detects.

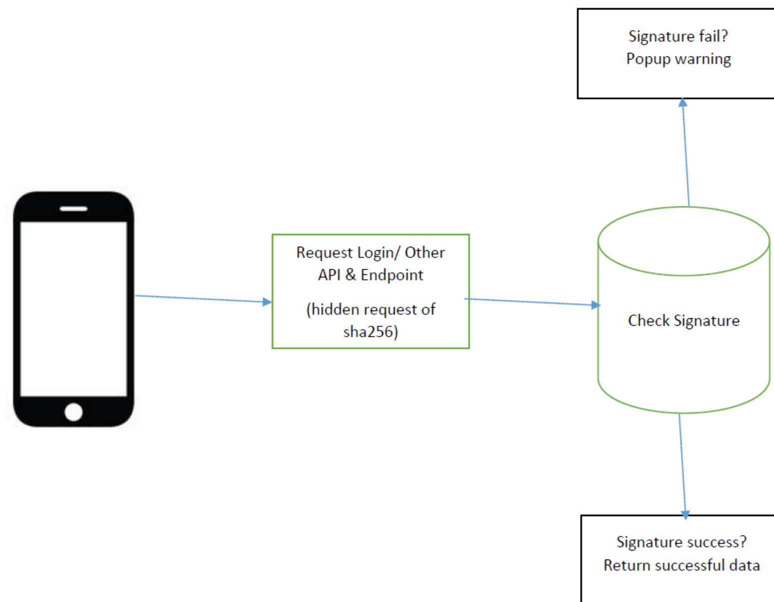

Figure 2: Scenario diagram when user using the applications

Figure 2 illustrates the scenario during the implementation to check whether the installed application is authentic by requesting signature from the application. If the application is fake, it will show a warning popup, hence users of the application are unable to use the apps.

Figure 3 : Scenario diagram Flutter framework requesting Android Native using Java for digital signatures

Figure 3 illustrates the scenario during the implementation to find the signatures by using flutter. Flutter framework will invoke method from java native code to get the digital signature of the current applications.

```
String signatures = "Uknown";
Future<void> _getSignatures() async {
  const platform = MethodChannel('test.fyp.myproject/getSignatures');
  try {
    var checkSignature = await platform.invokeMethod('checkSignature');

    if (kDebugMode) {
      print('checkSignature is $checkSignature');
    }
    setState(() => signatures = checkSignature);
  } on PlatformException catch (e) {
    if (kDebugMode) {
      print('Failed checkSignature is ${e.message}');
    }
  }
}
```

Figure 4 : Snippet code of calling Java library in Flutter using Dart language

Figure 4 illustrates the scenario during the implementation to find the installed application signatures by invoking a method that is written in Java codes by using Dart language.

```
private String checkSignature(){
    PackageInfo info = null;
    String signatures = null;
    try {
        info = getPackageManager().getPackageInfo(getPackageName(), PackageManager.GET_SIGNATURES);
    } catch (PackageManager.NameNotFoundException e) {
        e.printStackTrace();
    }

    if (null != info && info.signatures.length > 0) {
        byte[] rawCertJava = info.signatures[0].toByteArray();
        byte[] rawCertNative = bytesFromJNI();
        String str = "From Java:\n"+getInfoFromBytes(rawCertJava) + "From native:\n"+getInfoFromBytes(rawC
        // String str = getInfoFromBytes(rawCertJava);
        signatures = str;
    }
    return signatures;
}
```

Figure 5 : Snippet code of Java library using its own library to find the digital signature of the current application.

Figure 5 illustrates the snippet code and implementation of using native Android library in Java code to calculate and find the digital signatures.

```
Future myCustomImplementation(String url, Map<String, String> headers,
    List<String> allowedSHAFingerprints) async {
  print("url $url");
  try {
    final secure = await HttpCertificatePinning.check(
        serverURL: url,
        headerHttp: headers,
        sha: SHA.SHA256,
        allowedSHAFingerprints: allowedSHAFingerprints,
        timeout: 50);
    print("secure secure $secure");
    if (secure.contains("CONNECTION_SECURE")) {
      return secure;
    } else {
      return secure;
    }
  } catch (e) {
    return e.toString();
  }
```

Figure 6 : Code snippet of checking SSL signature in the mobile devices compared to the server URL

Figure 6 illustrates how the implementation for checking the project SSL certificate is allowed and the return URL certificate is the same as the one saved in the project. If the project and the URL have the same SSL certificate, hence it will return Connection Secure messages.

3.3 Implementation Phase

This phase is focused on implementing and setting up the environment. This project is using windows because Flutter Framework can support windows by using zip file download from the official Flutter website. Installation of VS Code will make it easier to code and implement the research.

4. Results and Discussion

The results and discussion section presents the results of the analysis and discussion based on the methodology used to develop the system and the collection of the data that has been conducted. From this point, the testing and development have been done successfully, and all of the results have been collected and analyzed.

Figure 7 : Results of the interface if the Signature is valid and SSL connection is secured

Figure 7 shows the results of the implementation. The Signed Signatures of both called from Java method and JNI (Java Native Interface) is the same, therefore it will check for the SSL signature to be the same with the server. The server used for testing is https://emirfikri.com. Therefore, it will check for the SSL Signature to be the same. When connection is secured, it will show the Connection Secured message.

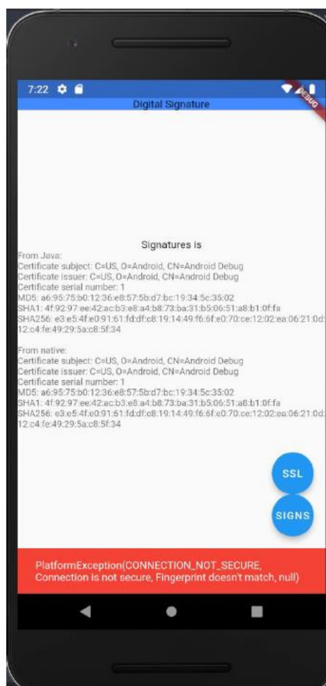

Figure 8 : Results of the interface if the Signature is valid and SSL connection is not secured

Figure 8 shows the results of the implementation. The Signed Signatures of both called from Java method and JNI (Java Native Interface) is the same, therefore it will check for the SSL signature to be

the same with the server. If the SSL return by the server is not the same as saved in the device application, it will abort the connection and shows, Connection Not Secure messages.

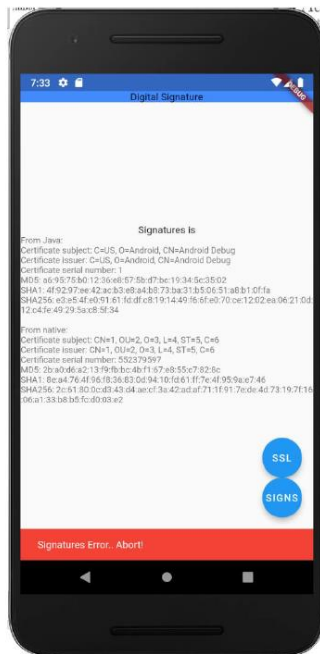

Figure 9 : Results of the interface if the Signature is invalid and SSL connection is secured

Figure 9 shows the results of the implementation. The Signed Signatures of both called from Java method and JNI (Java Native Interface) is the not the same, hence no connection will be made and can conclude that the application has been hijacked by hacker because the signature in Native Byte Code is different.

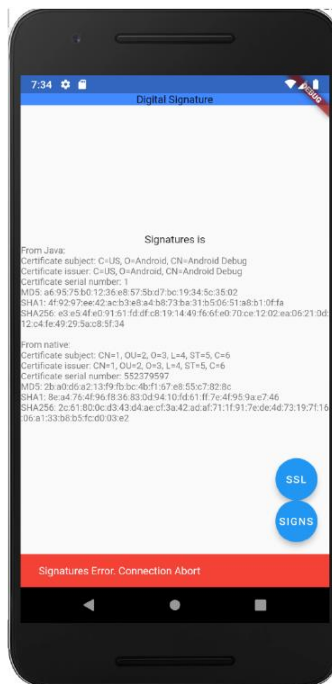

Figure 10 : Results of the interface if the Signature is invalid and SSL connection is not secured

Figure 10 shows the results of the implementation. The Signed Signatures is invalid; hence the application will not make any connection to the application server. To avoid the attacker, gain any information on the application.

5. Conclusion

In mobile application, it is crucial to get and know the application digital signatures, since it will ensure integrity, reliability of the applications to know either it is attacked or not. Hence the method of checking digital signature using Java method and using Java Native Interface that will call C/C++ library is implemented in order to avoid APK Signature Tampering. The main objective of the research is to design and develop a mobile application clone detection method using digital signature and tested the proposed mobile application clone detection method.

As extensive research, studies and implementation have been conducted in order to ensure success of this project. From the results and discussion in the chapter before, there are slight outcome and scenario were made to prove an outcome. However, throughout the studies and development of the project, even the Java library can be Tempered using Java APK Signature Killer, so it is good to be prepared in the future if the library has a loophole, need to be fixes.

Acknowledgment

The authors would like to thank the College of Computing, Informatic and Media, Universiti Teknologi MARA for its support.

References

[1] Baykara, Muhammet; Colak, Eren (2018). [IEEE 2018 6th International Symposium on Digital Forensic and Security (ISDFS) - Antalya (2018.3.22-2018.3.25)] 2018 6th International Symposium on Digital Forensic and Security (ISDFS) - A review of cloned mobile malware applications for android devices. , (), 1–5. doi:10.1109/ISDFS.2018.8355388

[2] Naeem, M. R., Ullah, F., Mostarda, L., & Shah, S. A. (2021). Clone detection in 5G-enabled social IoT system using graph semantics and deep learning model. International Journal of Machine Learning and Cybernetics, 12(11), 3115-3127. https://doi.org/10.1007/s13042-020-01246-9

[3] Charlie Soh, Hee Beng Kuan Tan, Yauhen Leanidavich Arnatovich, Lipo Wang, Detecting Clones in Android Applications through Analyzing User Interfaces, IEEE 23rd International Conference on Program Comprehension, 2015.

[4] Ashwag Albakri, Huda Fatima, Maram Mohammed, Aisha Ahmed, Aisha Ali, Asala Ali, Nahla Mohammed Elzein, "Survey on Reverse-Engineering Tools for Android Mobile Devices", Mathematical Problems in Engineering, vol. 2022, Article ID 4908134, 7 pages, 2022. https://doi.org/10.1155/2022/4908134.

[5] R. Sikder, S. Khan, S. Hossain, and W. Z. Khan, "A survey on android security: development and deployment hindrance and best practices," Telkomnika, vol. 18, no. 1, pp. 485–499, 2020.

[6] M. Protsenko, S. Kreuter and T. Müller, "Dynamic Self-Protection and Tamperproofing for Android Apps Using Native Code," 2015 10th International Conference on Availability, Reliability and Security, Toulouse, France, 2015, pp. 129-138, doi: 10.1109/ARES.2015.98.

[7] Duan, R., Bijlani, A., Ji, Y., Alrawi, O., Xiong, Y., Ike, M., Saltaformaggio, B., & Lee, W. (2019). Automating Patching of Vulnerable Open-Source Software Versions in Application Binaries. Proceedings 2019 Network and Distributed System Security Symposium.

[8] Mitropoulos, Dimitris & Spinellis, Diomidis. (2017). Fatal injection: A survey of modern code injection attack countermeasures. PeerJ Computer Science. 3. e136. 10.7717/peerj-cs.136.

[9] Hyunwoo Choi, Yongdae Kim, "Large-Scale Analysis of Remote Code Injection Attacks in Android Apps", Security and Communication Networks, vol. 2018, Article ID 2489214, 17 pages, 2018. https://doi.org/10.1155/2018/2489214

[10] Nurhaida, I. (2017). Digital Signature & Encryption Implementation For Increasing Authentication , Integrity , Security And Data Non-Repudiation.