Available online at http://icsejournal.com/index.php/JCSE
Journal of Computer Science an Engineering (JCSE)
e-ISSN 2721-0251
Vol. 5, No. 1, February 2024, pp. 46-63

Android Apps Vulnerability Detection with Static and Dynamic Analysis Approach using MOBSF

Sabrina Uhti Kusreynada[1], Azhari Shouni Barkah[2]

[1,2]Department of Information Technology, Faculty of Computer Science, Universitas Amikom Purwokerto, Purwokerto, Indonesia
uhti1811@gmail.com*,
* corresponding author

ARTICLE INFO

Article History:
Received January 25, 2023
Revised February 26, 2024
Accepted February 28, 2024

Keywords:
Mobile JKN Security
Cyber Threats in Healthcare
Application Vulnerabilities
SSL Pinning Analysis
Data Privacy Concerns

Correspondence:
E-mail: uhti1811@gmail.com

ABSTRACT

Ensuring the security of Android applications is paramount, especially for apps like Mobile JKN, launched by the Social Security Agency on Health "BPJS Kesehatan" under the Ministry of Health Republic Indonesia, which contain sensitive participant data. Such information is often targeted by cybercriminals seeking personal gain through data theft by exploiting security vulnerabilities within the application. To address these risks, a thorough analysis was conducted to detect security loopholes in the Mobile JKN application. The study used the Mobile Security Framework (MOBSF) tools and involved static and dynamic analyses. Despite the application's implementation of secure SSL Pinning and detection of rooted devices, the static analysis revealed potential security loopholes, including dangerous permission access, weak cryptographic methods, and vulnerable hardcoded secrets. Moreover, the application was found vulnerable to Janus, SQL Injection, and padding oracle attacks. While the dynamic analysis showed satisfactory implementation of SSL Pinning and no performance degradation, it also revealed that root detection was lacking, and debugger connections were not detected while the application was running. These findings emphasize the critical need for immediate security enhancements in the Mobile JKN application.

1. Introduction

Technological advancements have significantly increased the number of mobile applications in Indonesia. The ease with which these apps have simplified daily tasks has also sparked interest in their use. A diverse array of mobile applications, spanning social networking, e-government, and e-commerce, is now readily accessible. According to a report from Data.ai, in 2022, the average Indonesian spent 5.7 hours per day using mobile applications, marking a 5.56% increase from the previous year [1]. Along with the increasing use of mobile applications, there has also been an increase in the number of applications launched. Currently, the Android and iOS platforms have 21 million applications, with 77% being for Android. In Indonesia, more than 13,000 application downloads occurred every minute in 2021 [2]. The availability of various applications has driven user interest in utilizing them, resulting in increased efficiency across multiple sectors, including healthcare.

Public services in the healthcare sector are efforts undertaken by the government or other public institutions to fulfill the community's fundamental rights to healthcare. In carrying out its duties and responsibilities, the government must improve the effectiveness and efficiency of public services for the community [3]. One step the Indonesian government can take is to utilize information technology, particularly E-government. E-government is a system of information technology created by the government to improve public services by providing options for the public to access public information easily [4]. One implementation of E-government in the healthcare sector is Mobile JKN, the latest innovation from BPJS Kesehatan. This application facilitates online registration and provides access to information for participants. With over 10 million downloads on the Google Play

Store, this application allows users to view bills, obtain information about healthcare facilities, and submit suggestions or complaints [5].

Despite the continuous advancement of technology, data security poses a challenge that needs to be addressed. Threats such as data theft, malware attacks, or privacy breaches can jeopardize sensitive information stored within applications. In 2021, 279 million personal data records of Indonesian citizens who were participants of BPJS Kesehatan were suspected of having been leaked and sold online on the raid forum website, possibly due to application hacking [6]. Research by Amalia et al. found that the BPJS Kesehatan application did not use two-factor authentication, indicating weaknesses in its security level [7]. Therefore, evaluation with security analysis is necessary to detect vulnerabilities in the application.

According to Ibrar et al., there are three tools capable of automatically analyzing the security of mobile applications, namely MobSF (Mobile Security Framework), Androbugs Framework, and QARK (Quick Android Review Kit) [8]. However, QARK and AndroBugs Framework do not use specific approaches as a basis for testing application security [9]. Androbugs Framework focuses only on static analysis, while QARK focuses on dynamic analysis. MobSF, which can perform static and dynamic analysis, is a better choice.

Static analysis is used to check the application source code without having to run or test the application directly. Dynamic analysis examines the application while it is running. Because MobSF is open source, many researchers use it extensively to test application vulnerabilities [11]. Static analysis using MobSF has a high success rate, with an actual positive value of up to 97% [12].

Based on research conducted by Tansen and Nurdiarto, they analyzed the Bouncing Golf and Riltok Banking Trojan malware to observe the characteristics and behavior of these malware [13]. This research is important due to the proliferation of malware attacks by malicious application developers on the Android platform. Testing was conducted using static and dynamic analysis using the Mobile Security Framework (MobSF). The results showed that Bouncing Golf can steal information and control infected Android devices. Riltok Banking Trojan can also take control of smartphones to steal credit card information through phishing methods.

This study aims to conduct a security analysis of the Android-based BPJS Kesehatan application, Mobile JKN. This application contains sensitive information such as identity, medical history, bank account numbers, and other personal documents, making it an easy target for cyberattacks. This application will be analyzed statically and dynamically using the Mobile Security Framework. In static analysis, evaluations will be conducted on dangerous permissions, certificate analysis, manifest analysis, code analysis, and malware domain checks. Meanwhile, in dynamic analysis, evaluations will be conducted on API monitoring, SSL pinning bypass, root detection bypass, and debugger checker bypass.

2. Methods

2.1. Research Flow

The research flow represents a systematic collection and analysis process to address existing issues. In Figure 1, the initial stage of this research is observed to commence with problem identification, revealing a security gap concerning user data in Mobile JKN. Subsequently, a literature review is conducted, involving the search and comprehension of literature sources related to information security testing. Following the literature review, the next step is gathering tools and materials for application security testing. The subsequent process entails conducting security tests on Mobile JKN using the Mobile Security Framework to perform static and dynamic analyses. After completing tests using the Mobile Security Framework, the output will be a report. The final research stage involves concluding the application testing results.

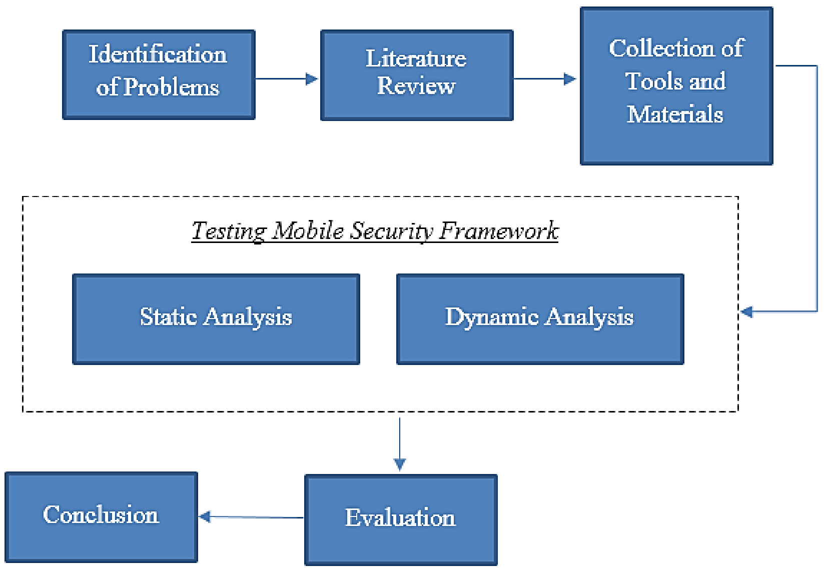

Identification of Problems
Literature Review
Collection of Tools and Materials
Testing Mobile Security Framework
Static Analysis
Dynamic Analysis
Conclusion
Evaluation

Figure 1. Research Flow

2.2. Mobile Security Framework

The Mobile Security Framework is a versatile tool designed to enhance mobile app security through its ability to conduct both static and dynamic analyses[14]. Static analysis involves examining an application's code and structure without executing it, aiming to identify potential vulnerabilities or security weaknesses. On the other hand, dynamic analysis consists of executing the application in a controlled environment to observe its behavior and identify security issues that may arise during runtime. This dual approach enables the framework to provide comprehensive insights into the security posture of mobile applications, helping developers and security professionals mitigate potential risks effectively.

Static Analyses

The Mobile Security Framework static analysis workflow consists of 3 processes: input, static analysis, and output, as seen in Figure 2. **Input:** The first process is to input data. In this research, the data input stage involves inserting APK extension application files. The Mobile Security Framework will analyze these files. **Static Analysis:** The static analysis in the Mobile Security Framework (MobSF) is used to identify potential issues in the analyzed files. This includes detecting insecure permissions and configurations that could lead to vulnerabilities, finding malicious code, and discovering changes in hidden file storage locations. In this static analysis stage, parameters are used to perform security tests on applications, including (i) Dangerous permissions are used to detect permissions that are dangerous to mobile devices. These permissions are considered dangerous because they can endanger the privacy or security of users if used irresponsibly. (ii) Certificate analysis in MobSF is used to check the authenticity and reliability of certificates used in the application. (iii) Manifest analysis performed by the Mobile Security Framework (MobSF) is the process of checking and evaluating the AndroidManifest.xml file in the Android application. (iv) Code Analysis checks and evaluates source code to identify potential defects, ensure compliance with coding standards, and improve software quality and security. This involves analyzing code changes, mapping them to programming language types, and using code analysis tools and analysis rules to execute analysis and obtain results. (v) Domain malware check checks and verifies whether the domains to be accessed are indicated as malware sources. **Output:** Using the Mobile Security Framework tool will provide detailed analysis results as a report. The report presents the results of the tests conducted using MobSF.

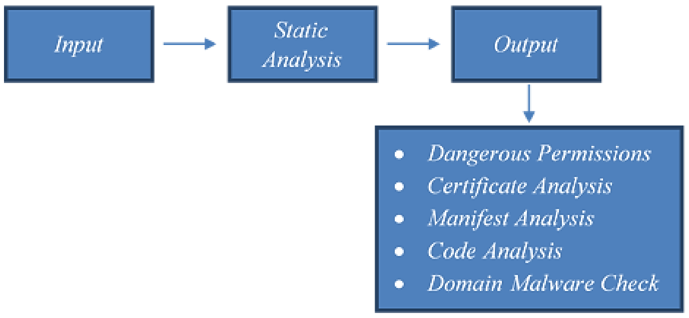


Figure 2. Workflow of Static Analysis

Dynamis Analyses

The workflow of dynamic analysis in the Mobile Security Framework consists of three processes: input, dynamic analysis, and output, as depicted in Figure 3. **Input:** The first step is data input. In this study, the data input stage involves inserting APK files. These files will undergo analysis by the Mobile Security Framework. **Dynamic Analysis:** The Mobile Security Framework (MobSF) inspects and analyzes the behavior and security of active applications or systems. This method involves executing the application or system in a controlled environment to examine how the application interacts with its environment and whether there are vulnerabilities that attackers can exploit. In this dynamic analysis stage, there are parameters used to test the security of the application, including the following: (i) API monitoring, which is the process of monitoring the APIs used in a system or application to ensure their availability, performance, and reliability in the production environment. API monitoring ensures the availability, performance, and reliability of APIs in the production environment. (ii) SSL Pinning Bypass can be used to identify and evaluate potential security vulnerabilities in SSL pinning implementations. Security testing can exploit vulnerabilities in the application's SSL/TLS certificate validation by bypassing SSL pinning. (iii) Root Detection Bypass, which can be used to identify vulnerabilities in the root detection mechanisms used by the application. By bypassing root detection, security testing can evaluate potential security vulnerabilities related to using rooted devices. (iv) Debugger Checker Bypass, which can be used to identify vulnerabilities in the debugger check mechanisms used by the application. By bypassing this, security testing can evaluate potential vulnerabilities related to using debuggers in the application. **Output:** Using the Mobile Security Framework tool will provide detailed analysis results in a report that includes information about the dynamic analysis results. The report presents the results of the tests performed using MobSF.

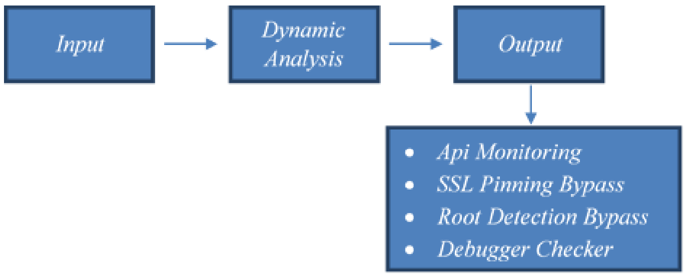


Figure 3. Workflow of Dynamic Analysis

3. Results and Discussion

The Mobile JKN application testing process is conducted using the Mobile Security Framework, which begins with static analysis to identify security vulnerabilities in the source code, followed by dynamic analysis by running the application on an emulator device. In static analysis, the testing parameters include Dangerous permission, Certificate analysis, Manifest analysis, Code Analysis, and Malware Domain Check. For dynamic analysis, the testing parameters include API Monitoring, SSL Pinning Bypass, Root Detection Bypass, and Debugger Checker Bypass. The following are the results of the static and dynamic analysis of the Mobile JKN application:

3.1. Static Analysis

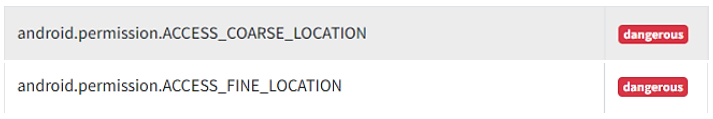
Static analysis is an approach used to detect and fix poor or vulnerable code by examining code listings, test results, or other documentation to identify errors, violations of development standards, or other issues to enhance system and software security. Below are the findings from static analysis using MobSF, including:

Dangerous Permission

The dangerous permissions testing parameter detects permissions that pose a dangerous risk in the Mobile JKN application. Access permissions in the application can be a security vulnerability if not managed properly or if abused. Here are the dangerous permissions in the Mobile JKN application detected by MobSF, including:

Location Access

Figure 5 shows that the Mobile JKN application has two dangerous permissions related to location access. These permissions allow accurate device location information to be obtained via cellular networks or Wi-Fi using GPS technology and other methods. Location access permissions are considered dangerous if the application requests them without clear justification or exploits them in ways that violate user privacy. With location access permissions in the Mobile JKN application, there is a potential security risk, including health information theft. Furthermore, user location data can be used for unrelated purposes, such as advertising or marketing analysis, leading to data misuse and violating user privacy.

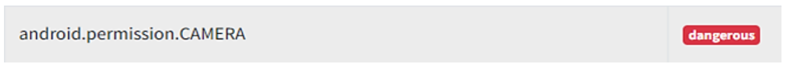


android.permission.ACCESS_COARSE_LOCATION	dangerous
android.permission.ACCESS_FINE_LOCATION	dangerous

Figure 5. Location Access in the Mobile JKN Application

Camera Access

In Figure 6, camera access permission is categorized as a dangerous permission. The image above displays the permissions the Android application requires to access the device’s camera. This permission enables the application to utilize the camera to capture photos or record videos through the user’s device, potentially affecting the user’s privacy. The camera access permission in Mobile JKN could be exploited by unauthorized parties for camera eavesdropping.

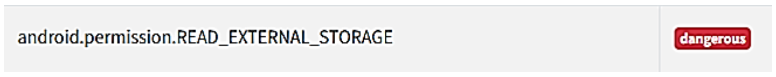


android.permission.CAMERA	dangerous

Figure 6. Camera Access in the Mobile JKN Application

Storage Data Reading

In Figure 7, the Mobile JKN application has access to external storage on the user’s device. This permission allows the application to read data such as images, videos, audio files, or other documents stored by the user in the device’s external storage.



android.permission.READ_EXTERNAL_STORAGE	dangerous

Figure 7. Data Storage Reading Access

Audio Recording Access

The Mobile JKN application has access to audio recording, as shown in Figure 8. This permission allows the application to request access to record audio using the user’s device microphone. This permission is used by various applications and systems to enable features such as voice recording, voice calls, and audio-based communication.

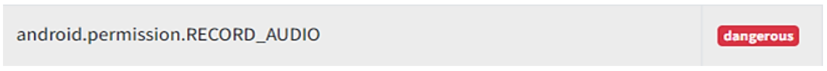

android.permission.RECORD_AUDIO dangerous

Figure 8. Audio Recording Access in the Mobile JKN Application

Data Writing Access

In addition to reading data on the user's device, the Mobile JKN application can also write data to the user's external storage, as shown in Figure 9. This permission allows the application to read, modify, and delete content in the user's external storage. To address dangerous access permissions in the application, ensure that it only requests and uses permissions necessary for its functionality and protect user privacy. Developers can take steps to address dangerous access permissions, including implementing the principle of least privilege by only requesting permissions necessary to perform the application's functionality. Additionally, use runtime permissions in the application, allowing users to grant access permissions only when needed.

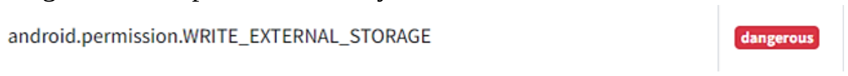
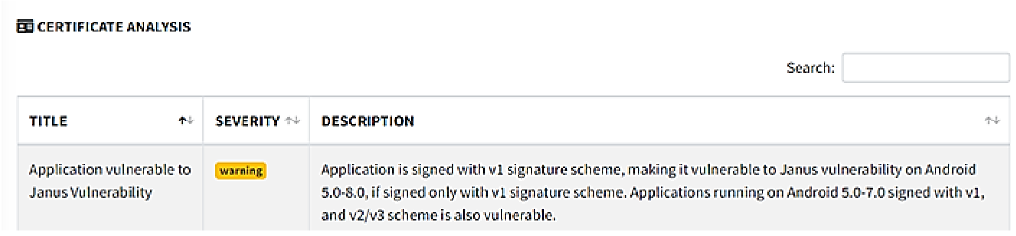

android.permission.WRITE_EXTERNAL_STORAGE dangerous

Figure 9. Writing Access to External Storage on User's Device

Certificate Analysis

Certificate analysis in MobSF is used to verify the authenticity and reliability of certificates used in the application. In this test, vulnerabilities were detected with a severity warning category. Figure 10 shows that the certificate analysis in the Mobile JKN application falls into the warning severity category. In this analysis, it was found that the Mobile JKN application is vulnerable to the Janus vulnerability. Janus is an attack that allows an attacker to view or gain access to sensitive data while maintaining legitimate or normal-looking access. In this case, the attacker can manipulate data sent or received by the system, often using encryption at higher or lower layers, such as communication protocols or data transmission. This way, attackers can inject malicious or damaging data into communications without being detected by the system or user.

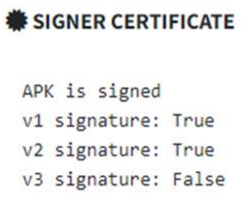

CERTIFICATE ANALYSIS

Search:

TITLE	SEVERITY	DESCRIPTION
Application vulnerable to Janus Vulnerability	warning	Application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability on Android 5.0-8.0, if signed only with v1 signature scheme. Applications running on Android 5.0-7.0 signed with v1, and v2/v3 scheme is also vulnerable.

Figure 10. Certificate Analysis in the Mobile JKN Application

Figure 11 shows the signature scheme used by Mobile JKN to secure the application. In the findings above, the Mobile JKN application is signed with signature schemes v1 and v2, making it vulnerable to the Janus vulnerability in Android versions 5.0 to 8.0. However, the Janus vulnerability may still exist even if the application uses signature schemes v1, v2, and v3. This is because the vulnerability lies in the v1 signature scheme. So, even when combined with v2 or v3 signature schemes, it may not fully address the security issue.


SIGNER CERTIFICATE

```
APK is signed
v1 signature: True
v2 signature: True
v3 signature: False
```

Figure 11. Signature Scheme Used by Mobile JKN

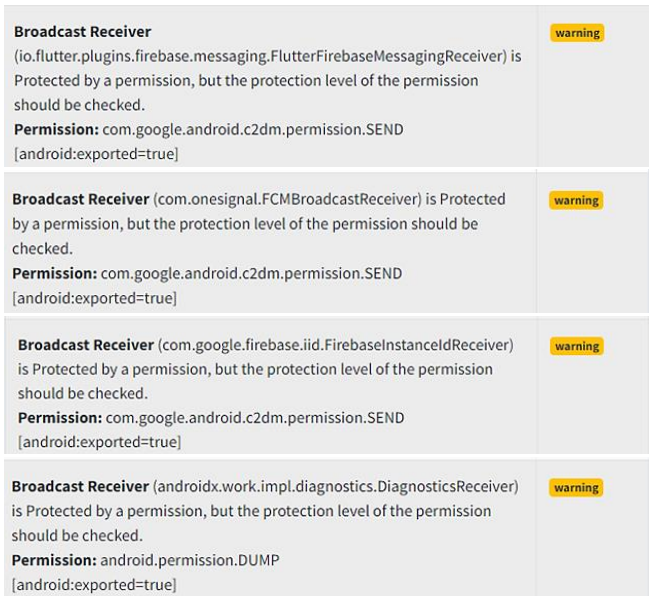
To address the Janus attack and protect the application from such security risks, developers can take steps to use a stronger signature scheme. They can use newer APK signature schemes, such as v2 or v3, with more vital security features than v1. The v2 and v3 schemes allow for multi-layered signing, meaning they can sign the APK with multiple private keys. This can help in more flexible development management and maintaining signature security. Additionally, the v2 or v3 schemes can prevent modification or insertion of undetected malicious code.

Manifest Analysis

The Manifest analysis conducted by the Mobile Security Framework (MobSF) inspects and evaluates the AndroidManifest.xml file in an Android application. In this test, several broadcast receivers and services were found to fall into the warning severity category, including:

Broadcast Receiver

A Broadcast Receiver is a component in the Android platform that allows applications to listen for and respond to events or messages sent by the system or other applications. It enables applications to respond to changes in the system environment or user interactions with the device, as seen in Figure 12.



Broadcast Receiver (io.flutter.plugins.firebase.messaging.FlutterFirebaseMessagingReceiver) is Protected by a permission, but the protection level of the permission should be checked.
Permission: com.google.android.c2dm.permission.SEND
[android:exported=true]

warning

Broadcast Receiver (com.onesignal.FCMBroadcastReceiver) is Protected by a permission, but the protection level of the permission should be checked.
Permission: com.google.android.c2dm.permission.SEND
[android:exported=true]

warning

Broadcast Receiver (com.google.firebase.iid.FirebaseInstanceIdReceiver) is Protected by a permission, but the protection level of the permission should be checked.
Permission: com.google.android.c2dm.permission.SEND
[android:exported=true]

warning

Broadcast Receiver (androidx.work.impl.diagnostics.DiagnosticsReceiver) is Protected by a permission, but the protection level of the permission should be checked.
Permission: android.permission.DUMP
[android:exported=true]

warning

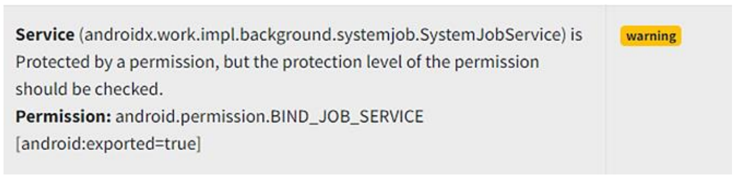
Figure 12. Warning Severity Receiver in the application

In Figure 12, four broadcast receivers are found in the Mobile JKN application. These broadcast receivers can exchange data with other applications on the user's device, making them accessible to any application. Permissions protect these broadcast receivers, but these permissions are not defined in the application. As a result, the level of protection from these permissions needs to be checked where the permissions are defined. In this case, there is a potential security risk because the shared Broadcast Receiver can be accessed by foreign applications that may have malicious intent. To mitigate this risk, developers can take a series of security measures to help protect the application from potential threats and security issues, such as using "LocalBroadcastManager" to limit broadcast reception only within the application. Additionally, make the appropriate Broadcast Receiver declarations in the application's manifest file by ensuring that broadcast reception only occurs when necessary.

Service in the Mobile JKN Application

Figure 13 shows a service in the Mobile JKN application categorized as warning severity. A Service is a component in an Android application that can run in the background without direct user interaction. This service can perform specific tasks, such as fetching data from the internet, processing information, or performing other actions. In this case, a service in the application is shared

with other applications on the same device. This means that other applications on the device can access this service. Based on the figure above, this service is protected by permissions that must be held by other applications that want to access it. These permissions prevent arbitrary applications from interacting with this service. However, these permissions are not defined within the application. In other words, the application does not have these permissions in its manifest file.
Suppose the permissions required by the component are not defined. In that case, it can lead to security issues that allow other applications or third parties to access features or resources quickly they should not have access to. To address security issues related to permissions not defined in an Android application, developers can take steps to declare the required permissions by ensuring that the permissions needed for the component, such as Activity, Service, or Broadcast Receiver, have been declared in the manifest file.


Service (androidx.work.impl.background.systemjob.SystemJobService) is Protected by a permission, but the protection level of the permission should be checked.
Permission: android.permission.BIND_JOB_SERVICE
[android:exported=true]

warning

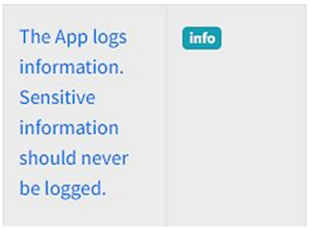
Figure 13. Warning Severity Service in the Mobile JKN Application

Code Analysis

Analysis of the code of the Mobile JKN application using MobSF revealed the results of the application testing, including :

App Log

Figure 14 shows the Mobile JKN application logs information. App logging is a record of events and actions that occur within an application and is used for various purposes, such as troubleshooting, user action identification, and system monitoring. This app logging can store sensitive information, so if this data falls into the wrong hands, it can lead to privacy breaches and security issues.

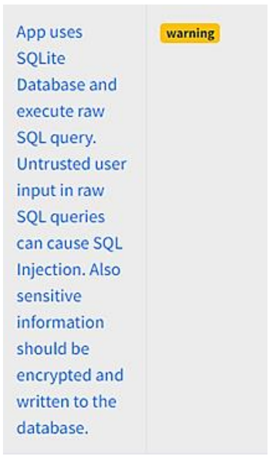

The App logs information. Sensitive information should never be logged.

info

Figure 14. Application Logging in Mobile JKN

SQLite Database

Figure 15 shows that the Mobile JKN application is detected to be using SQLite Database to store and manage information. SQLite database is a lightweight, self-contained relational database management system often used in desktop and mobile applications. In the figure above, the Mobile JKN application executes raw SQL queries, meaning it sends SQL queries directly to the database without using protection mechanisms or abstraction layers. As a result, user input that is not trusted can lead to SQL Injection attacks. SQL Injection is an attack that allows an attacker to inject malicious SQL code into queries executed by the database, which can damage or retrieve data from the database.


App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.

warning

Figure 15. The Mobile JKN Application Uses SQLite Database

Figure 16 illustrates the process of a SQL injection attack on the application. It depicts how an attacker identifies a security vulnerability in the application's input/login form and inputs a malicious SQL query. The database then validates and executes the query, granting the attacker access to view and retrieve data or act as a site administrator. To prevent SQL injection attacks, application developers can employ several strategies, such as using parameterized statements, validating user input, using prepared statements through the database API, escaping special characters, using stored procedures, and applying data sanitization.

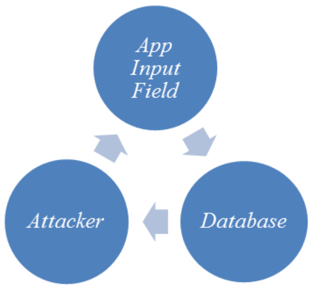
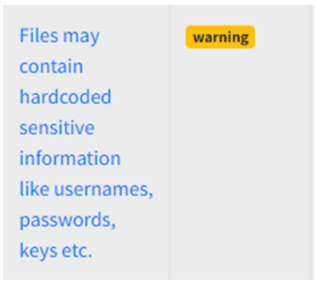


Figure 16. How SQL Injection Works

Hardcoded Secrets

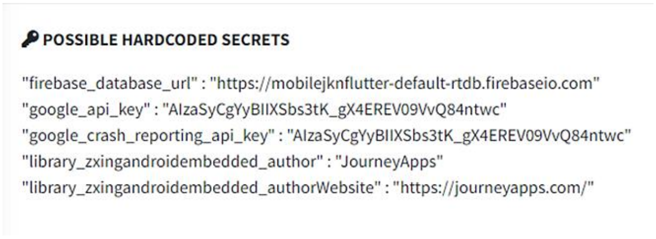
Figure 17 shows that the Mobile JKN application allows for loading files containing encoded sensitive information such as usernames, passwords, keys, and others. Hardcoded secrets refer to embedding confidential information, such as passwords, API keys, or other authentication information, directly into the application source code or configuration. Using hardcoded secrets can lead to serious security issues. If the code is leaked, copied, or shared, unauthorized parties can easily access sensitive information.


Files may contain hardcoded sensitive information like usernames, passwords, keys etc.

warning

Figure 17. Hardcoded Secrets in the Mobile JKN Application

Figure 18 shows possible hardcoded secrets in the Mobile JKN application detected by the Mobile Security Framework (MobSF). Figure 18 indicates that the configuration may contain hardcoded secrets. The figure above shows values found in the configuration file, which may contain sensitive information such as URLs, API keys, or other information that should not be publicly visible. To address potential hardcoded secrets in the application, developers can take several steps, such as using environment variables. Using environment variables is a highly recommended practice in software development for storing sensitive information and configuration that differs between development and production environments. Use a secure credential management system to store and manage sensitive information.


POSSIBLE HARDCODED SECRETS

```
"firebase_database_url" : "https://mobilejknflutter-default-rtdb.firebaseio.com"
"google_api_key" : "AIzaSyCgYyBIIXSbs3tK_gX4EREV09VvQ84ntwc"
"google_crash_reporting_api_key" : "AIzaSyCgYyBIIXSbs3tK_gX4EREV09VvQ84ntwc"
"library_zxingandroidembedded_author" : "JourneyApps"
"library_zxingandroidembedded_authorWebsite" : "https://journeyapps.com/"
```

Figure 18. Possible Hardcoded Secrets in Mobile JKN

Weak Crypto

Weak cryptography is the implementation of cryptographic techniques that are not strong or secure enough to protect data and communication from existing attacks and threats.

SHA-1

Figure 19 shows that the Mobile JKN application is detected to have weak cryptography, namely using SHA-1. SHA-1 is a weak hash function known to have hash collisions, making it vulnerable to differential cryptanalysis. This vulnerability makes it easier for attackers to create collision attacks, creating two different inputs that produce the same hash value, which can be used for malicious purposes[17].

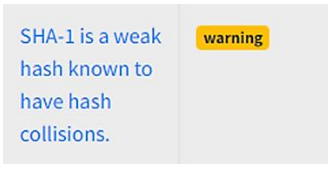

SHA-1 is a weak hash known to have hash collisions.	warning

Figure 19. The Use of SHA-1 in the Mobile JKN Application

In Figure 20, this code snippet uses the SHA-1 hash algorithm to generate a hash value from the given byte array. To avoid sophisticated cryptographic attacks, developers must use a strong hash function. Using the SHA-256 cryptographic hash function can provide an adequate level of security. SHA-256 is a well-known cryptographic hash algorithm widely used in many security applications [18].

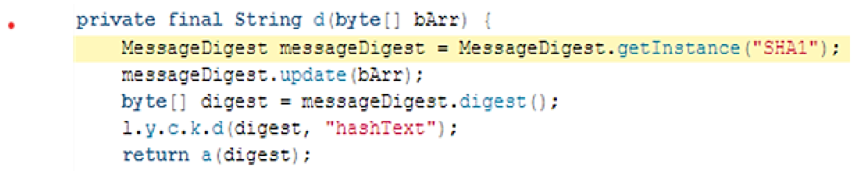
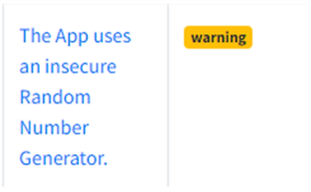

```
private final String d(byte[] bArr) {
    MessageDigest messageDigest = MessageDigest.getInstance("SHA1");
    messageDigest.update(bArr);
    byte[] digest = messageDigest.digest();
    l.y.c.k.d(digest, "hashText");
    return a(digest);
```

Figure 20. Code Using SHA-1

Random Number Generator

Figure 21 shows that the Mobile JKN application is detected to use an insecure random number generator. An insecure random number generator can lead to serious security issues in an application or system. Attackers can exploit the predictability of an insecure random number generator to predict the path or behavior of the system, opening up opportunities for sophisticated attacks. Some attacks, such as brute force or guessing attacks, can increase the chances of success if an insecure random number generator is used.

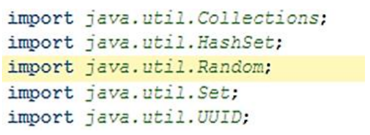


The App uses an insecure Random Number Generator.

warning

Figure 21. Insecure Random Number Generator

Figure 22 shows the use of an insecure random number generator in the Mobile JKN application. It is using the java.until.Random code is insufficient for cryptographic purposes or applications requiring high security. To create solid and secure random numbers, use java.security.SecureRandom for application security purposes. SecureRandom uses entropy sources from the operating system and algorithms designed to generate random numbers that are very difficult to predict.

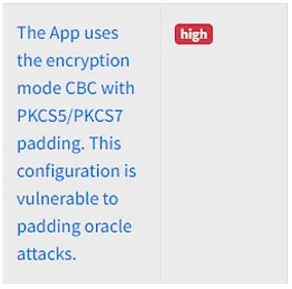


```
import java.util.Collections;
import java.util.HashSet;
import java.util.Random;
import java.util.Set;
import java.util.UUID;
```

Figure 22. Code Using Random Number Generator

CBC Encryption Mode

Figure 23 shows that the Mobile JKN application uses the CBC (Cipher Block Chaining) encryption mode with PKCS5/PKCS7 Padding. CBC encryption mode involves encrypting data block by block for security. PKCS5/PKCS7 Padding is a technique used to ensure that the encrypted data has a length suitable for the block size used by the encryption algorithm. However, this configuration is vulnerable to padding oracle attacks. This vulnerability arises due to encryption and padding operations that may result in security weaknesses. A padding oracle attack is a type of cryptographic attack that exploits the decryption process's behavior when handling invalid or incorrect padding in the ciphertext. This attack leverages information from the system's response to specific ciphertexts associated with padding errors. It is important to ensure correct and secure implementation of encryption modes and padding to avoid padding oracle attacks. Stronger encryption methods such as Authenticated Encryption with Associated Data (AEAD) can be considered for application security.



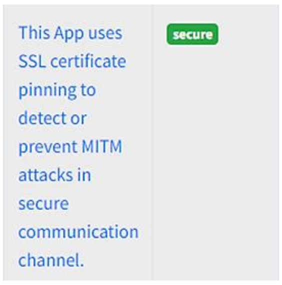
The App uses the encryption mode CBC with PKCS5/PKCS7 padding. This configuration is vulnerable to padding oracle attacks.

high

Figure 23. CBC Encryption Mode with PKCS5/PKCS7 Padding

SSL Pinning

Based on Figure 24, the Mobile JKN application uses SSL (Secure Socket Layer) certificate pinning to detect or prevent Man-in-the-Middle (MITM) attacks on secure communication channels. A Man-in-the-Middle (MITM) attack is a network attack aimed at violating the confidentiality and integrity of user data. This attack exploits vulnerabilities in the network infrastructure to intercept and manipulate data in transit. By using an SSL certificate, MITM attacks can be detected because changes or modifications to the SSL connection will result in a mismatch in the certificate received by the client.


This App uses SSL certificate pinning to detect or prevent MITM attacks in secure communication channel.

secure

Figure 24. SSL Pinning in Mobile JKN

Figure 25 shows that the code attempts to obtain the TrustManager related to SSL/TLS trust management, specifically X509TrustManager, which is used in configuring security connections.

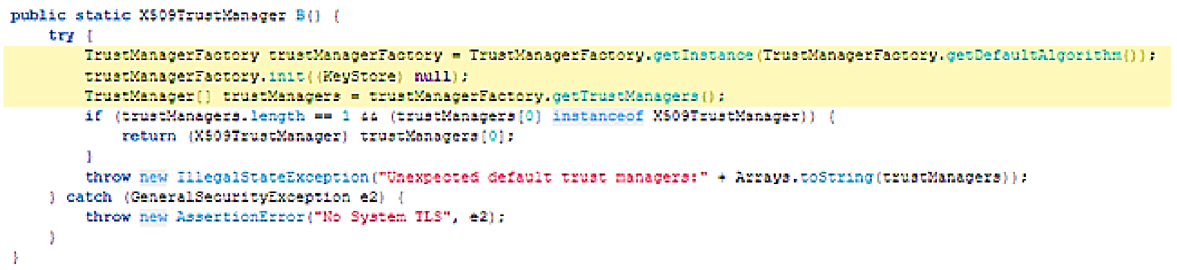
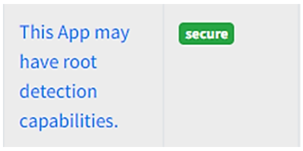

```java
public static X509TrustManager B() {
    try {
        TrustManagerFactory trustManagerFactory = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        trustManagerFactory.init((KeyStore) null);
        TrustManager[] trustManagers = trustManagerFactory.getTrustManagers();
        if (trustManagers.length == 1 && (trustManagers[0] instanceof X509TrustManager)) {
            return (X509TrustManager) trustManagers[0];
        }
        throw new IllegalStateException("Unexpected default trust managers:" + Arrays.toString(trustManagers));
    } catch (GeneralSecurityException e2) {
        throw new AssertionError("No System TLS", e2);
    }
}
```

Figure 25. The code in o/k0/e.java

Root Detection

Figure 26 shows that the Mobile JKN application may be able to detect whether the device running the application has been rooted. Rooting removes security restrictions imposed by the device manufacturer or operating system provider. This gives users higher access to the operating system and device components, allowing them to perform actions that are not accessible to non-rooted users, such as uninstalling system apps, modifying the operating system, and installing apps that require higher access.

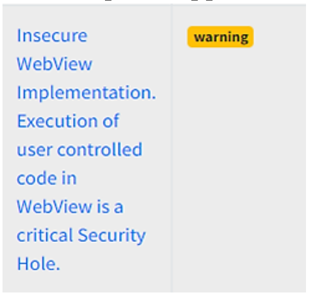

This App may have root detection capabilities.

secure

Figure 26. Root detection on the Mobile JKN application

WebView

The WebView is a component in Android applications that allows for rendering web content within the application itself without redirecting the user to a web browser. The test results in Figure 27 indicate that the implemented WebView is insecure. This means that if user-controlled code is executed within the WebView, it poses a critical security vulnerability. If users can input or control code within the WebView, attackers could exploit this to run malicious code, conduct hacking activities, steal sensitive information, or disrupt the application.


Insecure WebView Implementation. Execution of user controlled code in WebView is a critical Security Hole.

warning

Figure 27. The WebView in Mobile JKN

Aside from using an insecure WebView, the Mobile JKN application allows remote debugging on the WebView component. Figure 28 shows that the Mobile JKN application enables remote WebView debugging. Developers or security researchers can connect to the WebView from their device or computer externally to view, analyze, and troubleshoot HTML, JavaScript, CSS, and other web content displayed in the WebView. Although Remote WebView debugging is very useful for development and troubleshooting, it also has potential risks if enabled on applications released to the public or in a production environment. Enabling remote debugging can provide access to sensitive code and content in the application, which attackers can exploit if they manage to access it.

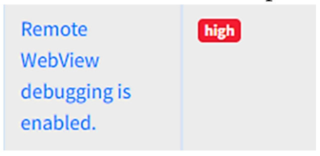
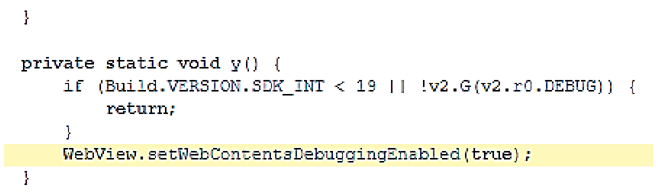

Remote WebView debugging is enabled. high

Figure 28. Remote WebView Debugging in the Mobile JKN application

Figure 29 shows that the Mobile JKN application enables remote debugging, which can pose a security risk. In developing applications intended for release, it is highly recommended to ensure that Remote WebView debugging is disabled by setting the option WebView.setWebContentsDebuggingEnabled(false) on the WebView in the application code.


```
}

private static void y() {
    if (Build.VERSION.SDK_INT < 19 || !v2.G(v2.r0.DEBUG)) {
        return;
    }
    WebView.setWebContentsDebuggingEnabled(true);
}
```

Figure 29. The code enabling remote debugging

Domain Malware Check

In static analysis, there is a check on the domain in the Mobile JKN application. The Malware Domain Check provides information about possible malware threats, vulnerabilities, or other malicious activities associated with the domain being checked. This helps identify and address security risks related to that domain. The results of the static analysis of the malware domain check are shown in Table 1.

Table 1. Malware Domain Check in the Mobile JKN application

Domain	Status
192.168.2.47	ok
aomedia.org	ok
api.flutter.dev	ok
api.onesignal.com	ok
api.whatsapp.com	ok
api3.qiscus.com	ok
apijkn.bpjs-kesehatan.go.id	ok
app-measurement.com	ok
developer.android.com	ok
developer.apple.com	ok
dvlp.bpjs-kesehatan.go.id	ok
ejkn.bpjs-kesehatan.go.id	ok
exoplayer.dev	ok
flutter.dev	ok
github.com	ok
goo.gl	ok
journeyapps.com	ok
mobilejknflutter-default-rtdb.firebaseio.com	ok
ns.adobe.com	ok
pagead2.googlesyndication.com	ok
play.google.com	ok
qc2.bpjs-kesehatan.go.id	ok

Domain	Status
realtime-lb.qiscus.com	ok
schemas.android.com	ok
schemas.microsoft.com	ok
sipp.bpjs-kesehatan.go.id	ok
webskrining.bpjs-kesehatan.go.id	ok
www.example.com	ok
www.google.com	ok
www.ibm.com	ok
www.w3.org	ok

Table 1. lists domains in the Mobile JKN application detected by MobSF. All domains detected by MobSF have an OK status, and no security potential has been detected. Therefore, the list of domains in the Mobile JKN application falls into the good category.

3.2. Dynamic analysis

Dynamic analysis is a technique used to inspect an application directly while it is running. In the security testing of the Mobile JKN application, the output results of dynamic analysis using MobSF with the testing parameters of API Monitoring, SSL Pinning Bypass, Root Detection Bypass, and Debugger Checker Bypass are displayed.

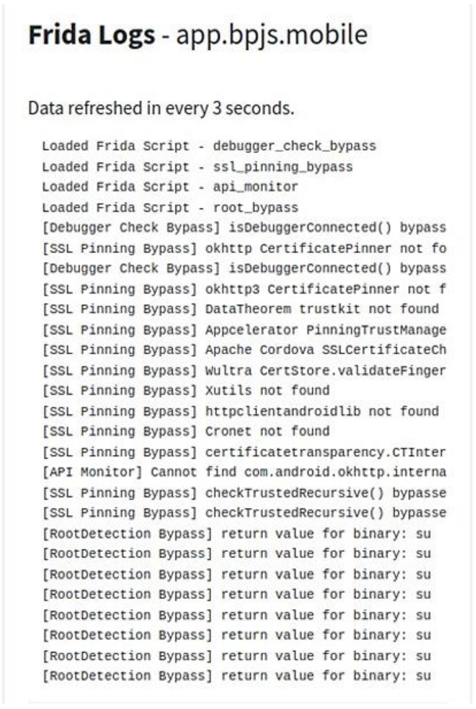
API Monitoring

Application Programming Interface (API) Monitoring refers to collecting and examining data related to the functionality of an API, which is responsible for enabling communication between various software applications. The goal of this process is to identify the impact experienced by users when leveraging the API. Implementing an application that utilizes services and libraries in its development can affect the application's overall performance. The API Monitoring process examines several activities within the application, as observed in the testing conducted on the Mobile JKN application using mobile security frameworks. The test results indicate no evidence of performance weaknesses in the application programming interface of the Mobile JKN application. Therefore, it can be concluded that the API monitoring process is crucial to ensuring smooth application functionality.

SSL Pinning Bypass

SSL Pinning Bypass is used to validate the SSL certificate from the target server. SSL pinning in applications ensures that only certificates deemed valid or pre-defined will be trusted. Figure 30 shows the SSL Pinning Bypass testing log against the Mobile JKN application. In this test, no security vulnerabilities were found during the SSL pinning bypass process that could threaten the security of the Mobile JKN application. This indicates that the developers have implemented SSL Pinning to ensure that only connections to servers with specific SSL certificates are accepted and to prevent interception of traffic between the client and server using fake certificates.

e-ISSN 2721-0251



Frida Logs - app.bpjs.mobile

Data refreshed in every 3 seconds.

```
Loaded Frida Script - debugger_check_bypass
Loaded Frida Script - ssl_pinning_bypass
Loaded Frida Script - api_monitor
Loaded Frida Script - root_bypass
[Debugger Check Bypass] isDebuggerConnected() bypass
[SSL Pinning Bypass] okhttp CertificatePinner not fo
[Debugger Check Bypass] isDebuggerConnected() bypass
[SSL Pinning Bypass] okhttp3 CertificatePinner not f
[SSL Pinning Bypass] DataTheorem trustkit not found
[SSL Pinning Bypass] Appcelerator PinningTrustManage
[SSL Pinning Bypass] Apache Cordova SSLCertificateCh
[SSL Pinning Bypass] Wultra CertStore.validateFinger
[SSL Pinning Bypass] Xutils not found
[SSL Pinning Bypass] httpclientandroidlib not found
[SSL Pinning Bypass] Cronet not found
[SSL Pinning Bypass] certificatetransparency.CTInter
[API Monitor] Cannot find com.android.okhttp.interna
[SSL Pinning Bypass] checkTrustedRecursive() bypasse
[SSL Pinning Bypass] checkTrustedRecursive() bypasse
[RootDetection Bypass] return value for binary: su
[RootDetection Bypass] return value for binary: su
[RootDetection Bypass] return value for binary: su
[RootDetection Bypass] return value for binary: su
[RootDetection Bypass] return value for binary: su
[RootDetection Bypass] return value for binary: su
[RootDetection Bypass] return value for binary: su
[RootDetection Bypass] return value for binary: su
```

Figure 30. SSL Pinning Bypass Log on Mobile JKN

Root Detection Bypass

Root detection bypass is an attempt to avoid the detection of root access or superuser permissions on Android devices, which can access and modify the system entirely. Figure 31 shows the superuser application in the Mobile JKN application. The image above indicates that the application does not detect requests for superuser permissions from other applications. This means the application does not detect root access, even though the device has been rooted.

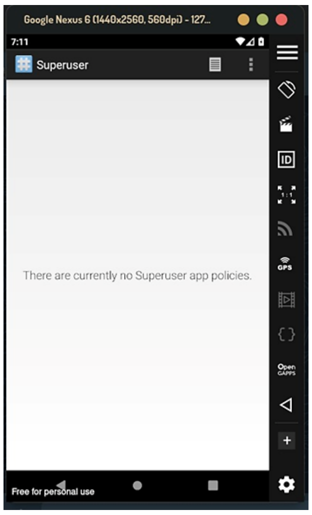
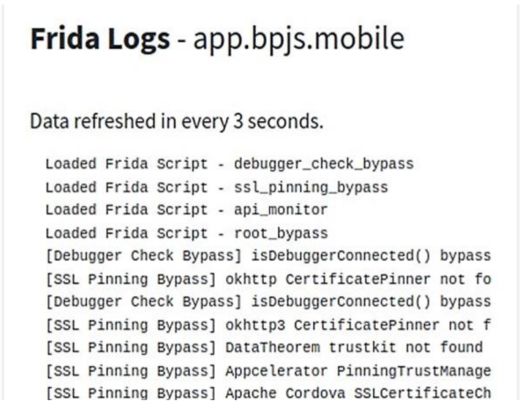

Google Nexus 6 (1440x2560, 560dpi) - 127...
7:11
Superuser
There are currently no Superuser app policies.
Free for personal use

Figure 31. The Superuser application in Mobile JKN

Debugger Checker Bypass

Debugger checker bypass is a technique used to detect the presence of a debugger in an application or software. A debugger is a tool used to inspect, monitor, and analyze the execution of code within an application.


Frida Logs - app.bpjs.mobile

Data refreshed in every 3 seconds.

```
Loaded Frida Script - debugger_check_bypass
Loaded Frida Script - ssl_pinning_bypass
Loaded Frida Script - api_monitor
Loaded Frida Script - root_bypass
[Debugger Check Bypass] isDebuggerConnected() bypass
[SSL Pinning Bypass] okhttp CertificatePinner not fo
[Debugger Check Bypass] isDebuggerConnected() bypass
[SSL Pinning Bypass] okhttp3 CertificatePinner not f
[SSL Pinning Bypass] DataTheorem trustkit not found
[SSL Pinning Bypass] Appcelerator PinningTrustManage
[SSL Pinning Bypass] Apache Cordova SSLCertificateCh
```

Figure 32. Log Debugger Checker Bypass pada Mobile JKN

Figure 32 shows the debugger checker bypass log on the Mobile JKN application when analyzed dynamically. The Mobile JKN application does not detect any connected debugger based on the image above. If no debugger is detected connected to the application while running, developers will have difficulty tracking, analyzing, and fixing any issues that may arise. Debugging becomes much more difficult because developers will not have the ability to check variable values, see execution flow, or find points where problems occur.

3.3. Static and Dynamic Analysis Results

The security assessment of the Mobile JKN application was conducted using both static and dynamic analysis techniques. The static analysis results, presented in Table 2, reveal several vulnerabilities and security issues within the application's code and permissions. Table 3, on the other hand, showcases the dynamic analysis results, highlighting the application's performance and its ability to resist certain security bypass techniques.

Table 2. Static analysis results

No	Test Parameters	Results
1.	*Dangerous Permissions*	The Mobile JKN application was found to have security vulnerabilities in the permission access section. Five access permissions were categorized as dangerous severity, including location access, camera access, reading data storage, audio recording access, and data writing access. This is because requesting access permissions without a clear purpose can violate user privacy.
2.	*Certificate Analysis*	The Mobile JKN application is vulnerable to Janus attacks. This is because the application uses signature schemes v1 and v2 to secure the application.
3.	*Manifest Analysis*	The Mobile JKN application was detected to be able to exchange data with other applications on the same user device.
4.	*Code Analysis*	In the code analysis, several findings were identified: – *App Log: The Mobile JKN application can log information. However, sensitive information should not be logged.* – *SQLite Database: Mobile JKN sends queries directly (raw SQL) to the database without using protection mechanisms. This makes the application vulnerable to SQL injection attacks.* – *Hardcoded Secrets: in the Mobile JKN application, sensitive information is directly inserted into the application's source code.* – *Weak Crypto: The Mobile JKN application uses weak cryptography to secure data. The application was detected using the SHA-1 hash function, a weak random number generator, and CBC encryption mode.* – *SSL Pinning: The Mobile JKN application uses secure SSL certificates.* – *Root Detection: The Mobile JKN application can detect rooted devices.* – *WebView: The Mobile JKN application uses an insecure WebView. Additionally, the application enables remote WebView debugging, which can pose a security risk.* – *In the Mobile JKN application, all detected domains have an ok status and are categorized as good.*

No	Test Parameters	Results
5.	*Domain Malware Check*	In the Mobile JKN application, all detected domains have an "ok" status and fall into the "good" category.

Table 3. Results of Dynamic Analysis

No	Testing Parameters	Results
1.	API Monitoring	When running the Mobile JKN application, no performance degradation is observed.
2.	SLL Pinning Bypass	The Mobile JKN application implements secure SSL Pinning.
3.	Root Detection Bypass	The Mobile JKN application is not yet able to fully detect rooted devices.
4.	Debugger Checker Bypass	The Mobile JKN application does not detect any connected debuggers, making it difficult for developers to track, analyze, and fix any issues that may arise in the application.

Based on the static and dynamic analysis findings as presented in Tables 2 and 3, the Mobile JKN application exhibits a mix of strengths and vulnerabilities. While the application demonstrates robust SSL Pinning and root detection capabilities, it also reveals permissions, certificate usage, code security, and WebView implementation vulnerabilities. These vulnerabilities pose a significant risk to user privacy and application security, underscoring the urgent need for security enhancements. Additionally, while the application shows satisfactory performance and effective SSL Pinning implementation, improvements are necessary to enhance its ability to detect rooted devices and debugger connections, bolstering overall security and developer capabilities.

4. Conclusion

The static analysis results indicate that the application does not contain malicious programs in the malware domain check, can detect rooted devices, and uses secure SSL Pinning. However, several potential security vulnerabilities were also found, such as the detection of dangerous access permissions, the use of weak cryptography methods, the presence of services, activities, and the use of vulnerable hardcoded secrets. In addition, the application was found to be vulnerable to Janus attacks, SQL Injection, and padding oracle attacks. Meanwhile, the dynamic analysis results show that the application has implemented SSL Pinning to secure communication pathways, and no performance degradation was observed during the test activities. However, when root detection bypass was performed during dynamic analysis, it was found that root detection was not implemented in the application, and it did not detect any connected debugger while the application was running.

References

[1] R. Mustajab, " Durasi bermain aplikasi mobile di Indonesia meningkat pada 2022," retrieved from https://dataindonesia.id/digital/detail/durasi-bermain-aplikasi-mobile-di-indonesia-meningkat-pada-2022, 2023, accessed on June 21, 2023.

[2] Data.ai, "State of Mobile 2022 Indonesia," retrieved from https://www.data.ai/en/go/state-of-mobile-2022-indonesia, 2022, accessed on June 20, 2023.

[3] S. Solechan, "Badan Penyelenggara Jaminan Sosial (BPJS) Kesehatan Sebagai Pelayanan Publik," Administrative Law and Governance Journal, vol. 2, no. 4, pp. 686-696, Nov. 2019. doi: 10.14710/alj.v2i4.686-696

[4] V. Wirawan, "Penerapan E-Government dalam Menyongsong Era Revolusi Industri 4.0 Kontemporer di Indonesia," Jurnal Penegakan Hukum dan Keadilan, vol. 1, no. 1. Universitas Muhammadiyah Yogyakarta, 2020. doi: 10.18196/jphk.1101.

[5] A. Wulanadary, S. Sudarman, and I. Ikhsan, "Inovasi Bpjs Kesehatan Dalam Pemberian Layanan Kepada Masyarakat : Aplikasi Mobile Jkn," Jurnal Public Policy, vol. 5, no. 2. Universitas Teuku Umar, p. 98, Oct. 31, 2019. doi: 10.35308/jpp.v5i2.1119.

[6] R. Ratra and P. Gulia, "Privacy Preserving Data Mining: Techniques and Algorithms," International Journal of Engineering Trends and Technology, vol. 68, no. 11. Seventh Sense Research Group Journals, pp. 56–62, Nov. 25, 2020. doi: 10.14445/22315381/ijett-v68i11p207

[7] R. Amalia, Wasilah, and Rini Nurlistiani, "Evaluasi dan Audit Aplikasi Mobile JKN pada BPJS Kesehatan Menggunakan Model TAM dan COBIT 5.0", JUPITER, vol. 14, no. 2-a, pp. 157–166, Oct. 2022. doi: 10.5281./4734/5.jupiter.2022.10

[8] F. Ibrar, H. Saleem, S. Castle, and M. Z. Malik, "A Study of Static Analysis Tools to Detect Vulnerabilities of Branchless Banking Applications in Developing Countries," Proceedings of the Ninth

International Conference on Information and Communication Technologies and Development. ACM, Nov. 16, 2017. doi: 10.1145/3136560.3136595.

[9] M. Antonishyn, “Mobile applications vulnerabilities testing model,” Collection “Information Technology and Security,” vol. 8, no. 1. Kyiv Politechnic Institute, pp. 49–57, Jul. 09, 2020. doi: 10.20535/2411-1031.2020.8.1.218003.

[10] B. Yankson, J. V. K, P. C. K. Hung, F. Iqbal and L. Ali, “Security Assessment for Zenbo Robot Using Drozer and mobSF Frameworks,” 2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 2021, pp. 1-7, doi: 10.1109/NTMS49979.2021.9432666.

[11] H. Shahriar, C. Zhang, M. A. Talukder, and S. Islam, “Mobile Application Security Using Static and Dynamic Analysis,” Studies in Computational Intelligence. Springer International Publishing, pp. 443–459, Dec. 15, 2020. doi: 10.1007/978-3-030-57024-8_20.

[12] T. H. Chiboora, L. Chacha, T. Byagutangaza, and A. Gueye, “Evaluating Mobile Banking Application Security Posture Using the OWASP’s MASVS Framework,” Proceedings of the 6th ACM SIGCAS/SIGCHI Conference on Computing and Sustainable Societies. ACM, Aug. 16, 2023. doi: 10.1145/3588001.3609367.

[13] B. Bokolo, G. Sur, Q. Liu, F. Yuan and F. Liang, “Hybrid Analysis Based Cross Inspection Framework for Android Malware Detection,” 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), Las Vegas, NV, USA, 2022, pp. 99-105, doi: 10.1109/SERA54885.2022.9806746.

[14] M. S. Rahman, B. Kojusner, R. Kennedy, P. Pathak, L. Qi and B. Williams, “So {U} R CERER: Developer-Driven Security Testing Framework for Android Apps,” 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), Melbourne, Australia, 2021, pp. 40-46, doi: 10.1109/ASEW52652.2021.00020.

[15] Shahriar, Hossain, Md Arabin Talukder, and Md Saiful Islam. “An exploratory analysis of mobile security tools.” (2019). doi: 10.1080/19393555.2020.1741743.

[16] T. Mantoro, D. Stephen and W. Wandy, “Malware Detection with Obfuscation Techniques on Android Using Dynamic Analysis,” 2022 IEEE 8th International Conference on Computing, Engineering and Design (ICCED), Sukasbumi, Indonesia, 2022, pp. 1-6, doi: 10.1109/ICCED56140.2022.10010359.

[17] A. Bakhtiyor, A. Orif, B. Ilkhom and K. Zarif, “Differential Collisions in SHA-1,” 2020 International Conference on Information Science and Communications Technologies (ICISCT), Tashkent, Uzbekistan, 2020, pp. 1-5, doi: 10.1109/ICISCT50599.2020.9351441.

[18] B. Kieu-Do-Nguyen, T. -T. Hoang, C. -K. Pham and C. Pham-Quoc, “A Power-efficient Implementation of SHA-256 Hash Function for Embedded Applications,” 2021 International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh City, Vietnam, 2021, pp. 39-44, doi: 10.1109/ATC52653.2021.9598264.