MOBILE PENTEST REPORT
READEASE MOBILE APPLICATION PENTEST REPORT

Group Members:
Garrick Julianta Arihta - 2802420435
Joseph Elnathan Wibowo - 2802420441
Mario Jovanka - 2802420795

Jakarta, June 20th, 2026

Table of Contents

1. Executive Summary

We did mobile application penetration testing on an Android application called ReadEase (package: com.ease.newreading), an entertainment-focused eBook platform offering genres such as werewolf fiction, romance, action, and adventure. The goal of this engagement was to identify security vulnerabilities in the application's local data storage, runtime screen-capture protections, and handling of user/device identifiers.

Throughout the testing, we used both static and dynamic analysis techniques. On the static side, we decompiled the application using JADX to inspect its source code, the AndroidManifest.xml file, and other files later used for information gathering, and analyzed its local storage via a rooted ADB shell. On the dynamic side, we ran the application on a rooted emulator using Android Studio and used Frida for runtime instrumentation to observe and manipulate application behavior. Our testing methodology was aligned with the OWASP Mobile Application Security Verification Standard (MASVS), allowing us to map each finding to a relevant MASVS category.

In total, we identified three confirmed vulnerabilities, one High and two Medium severity under the CVSS 4.0 scoring system, each targeting a different layer of the application, plus two additional hardcoded third-party credentials flagged as observations for remediation. To address these findings, we recommend encrypting all sensitive data before it is stored on the device, treating client-side screen-capture protection as only one layer of defense rather than the sole control, and encrypting device identifiers before they are passed off the device. Detailed recommendations for each finding are outlined in the sections below.

1.1 Testing Summary

Testing was performed against the publicly available release of ReadEase (version 2.0.5, developed by Bersama Sahabat) using a combination of static and dynamic analysis and testing. The assessment covered the application's local shared-preferences storage, its screen-capture protection logic, and its handling of the Google Advertising ID, in addition to a review of hardcoded credentials bundled with the app.

1.2 Recommendation

H1: Insecure Local Storage of Authentication Token and Session Data

The application currently stores the user's authentication token and session flags in plaintext on the device. We recommend encrypting all sensitive values before they are persisted, using EncryptedSharedPreferences or the Android Keystore, and shortening token lifetimes with refresh-token rotation.

M1: FLAG_SECURE Bypass Enabling Unauthorized Screenshot and Screen Recording

The app's only protection against screen capture of premium content is a client-side flag that can be disabled during runtime with tools such as Frida. We recommend treating FLAG_SECURE as one layer among several security layers, pairing it with server-side session checks, and instrumentation detection.

M2: Unencrypted Transmission of Advertising ID (ADID).

The device's advertising identifier is sent without application-layer encryption and without a clear user-facing notice. We recommend encrypting the identifier before transmission and providing transparent consent and privacy-policy disclosures.

2. Overview

ReadEase is an entertainment-focused mobile application offering access to a wide library of eBooks across genres including werewolf fiction, romance, action, and adventure, allowing users to discover, read, and enjoy digital stories through a mobile reading experience. The application is published on the Google Play Store under the package name com.ease.newreading, is rated for ages 16+ (strong language, in-app purchases), and is developed by the developer Bersama Sahabat. At the time of testing, the app was at version 2.0.5 (updated January 20, 2026), had surpassed 10,000+ downloads, required Android 7.0 or higher, and offered in-app purchases ranging from Rp 16,000 to Rp 1,690,000 per item. The application is also monetized in part through embedded advertising SDKs (including AppLovin and the Facebook Audience Network).

2.1 Methodology

To effectively gather information (information gathering) and assess the application for weaknesses, we combined both static and dynamic analysis techniques:

Static Analysis:

- APK Decompilation: Decompiling the application using JADX to inspect source code and application logic.
- Manifest & Permission Review: Analyzing AndroidManifest.xml for permissions, exported components, and security misconfigurations.
- Sensitive Data Inspection: Reviewing SharedPreferences, local databases, and configuration files for exposed credentials and tokens.
- Code & API Analysis: Identifying hardcoded secrets, API endpoints, and insecure implementation patterns.

Dynamic Analysis:

- Runtime Traffic Monitoring: Observing network communication between the application and backend services.
- Authentication Testing: Evaluating login mechanisms, session handling, and token management during execution.
- Runtime Instrumentation: Using Frida to monitor application behavior and bypass client-side security controls for testing purposes.
- Security Validation: Verifying discovered vulnerabilities on a live, rooted Android emulator and assessing their real-world impact.

The toolset used during our testing included JADX for static decompilation, the Android Debug Bridge (ADB) for direct shell access to the test device's file system, an Android emulator (API level matching Android 15) to host the application under test, and Frida for dynamic instrumentation and runtime hooking of Java methods.

2.2 Scope

The scope of this penetration test is specifically focused on the ReadEase Android application, its local data storage, and its immediate API communications.

Target / Asset	Description
ReadEase Mobile Application	Package name com.ease.newreading. The primary Android application binary (APK), built on the Flutter framework, and its internal logic.
Local Data Storage	Shared preference XML files located at /data/data/com.ease.newreading/shared_prefs/, primarily FlutterSharedPreferences.xml, which stores authentication and session state.
Runtime Screen-Capture Controls	FLAG_SECURE implementation (disableScreenshots() / allowScreenshots() methods) intended to protect premium book content from screenshots and screen recording.
Advertising / Device Identifier Handling	Logic responsible for retrieving and transmitting the Google Advertising ID (ADID), including the getGoogleADID method and its associated network call.

3. Findings Severity Ratings

ID	CVSS	Description	MASVS ID
H1	7.1	Insecure Local Storage of Authentication Token and Session Data	MASVS-STORAGE
M1	6.8	FLAG_SECURE Bypass Enabling Unauthorized Screenshot / Screen Recording	MASVS-RESILIENCE
M2	5.3	Unencrypted Transmission of Advertising ID (ADID)	MASVS-NETWORK / MASVS-PRIVACY

We identified a total of three vulnerabilities within the application, consisting of one High and two Medium severity findings based on the CVSS 4.0 scoring system, in addition to two hardcoded-credential observations. The High-severity finding relates to the insecure, plaintext storage of authentication and session data on the device. The first Medium-severity finding concerns the absence of protection against dynamic analysis and runtime instrumentation, which allows an attacker to bypass the app's client-side screenshot and screen-recording protection. The second Medium-severity finding relates to the unencrypted transmission of the device's advertising identifier.

4. Technical Findings

H1: Insecure Local Storage of Authentication Token and Session Data	
Score	7.1 (High)
Vector String	CVSS:4.0/AV:L/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N
Target	/data/data/com.ease.newreading/shared_prefs/FlutterSharedPreferences.xml

Overview

The ReadEase application persists sensitive authentication and session-state data in plaintext inside its Flutter shared preferences file. Because Android's SharedPreferences mechanism stores data as a readable XML file inside the application's private data directory, anyone with access to that directory, for example on a rooted device, through a device backup, or via another vulnerability granting file-system access can read the user's authentication token and impersonate them without needing their password. This finding maps to OWASP MASVS-STORAGE, which requires sensitive data to be stored using platform-appropriate, encrypted mechanisms.

Details

Using ADB, we obtained root shell access to the emulator and navigated to the application's private data directory to enumerate its shared preferences files:

```
C:\Users\LENOVO>adb shell
emu64xa:/ $ su
emu64xa:/ # cd data/data
emu64xa:/data/data # cd com.ease.newreading
emu64xa:/data/data/com.ease.newreading # cd shared_prefs/
emu64xa:/data/data/com.ease.newreading/shared_prefs # ls
AwOriginVisitLoggerPrefs.xml
DEBUG_PREF.xml
FBAdPrefs.xml
FlutterSharedPreferences.xml
Mediation_Shared_Preferences.xml
WebViewChromiumPrefs.xml
com.applovin.sdk.1.xml
com.applovin.sdk.preferences.wUsIp-ZWfrE3T4y8XB9GlBt4CiaTBtScv5lpC8sO3GrFTfTSZdOUxOXAh0374fh5QleV2oTtTMteIS08ldciNs.xml
com.applovin.sdk.shared.xml
com.ease.newreading_preferences.xml
com.facebook.ads.AD_REPORTING_CONFIG.xml
com.facebook.ads.FEATURE_CONFIG.xml
com.facebook.ads.LOCAL_COUNTERS.xml
com.facebook.ads.flash.xml
com.facebook.ads.idfa.xml
com.facebook.ads.internal.btextras.xml
com.facebook.ads.internal.ua.xml
com.facebook.internal.MODEL_STORE.xml
com.facebook.internal.preferences.APP_GATEKEEPERS.xml
com.facebook.internal.preferences.APP_SETTINGS.xml
com.facebook.sdk.USER_SETTINGS.xml
```

H1.1 – Shared preferences directory listing via rooted ADB shell

Using ADB, we obtained root shell access to the emulator and navigated to the application's private data directory to enumerate its shared preferences files:

```
<boolean name="flutter.continueRead" value="false" />
<string name="flutter.lastOpenDate">2026-05-24</string>
<long name="flutter.sourceType" value="1" />
<boolean name="flutter.openApp" value="true" />
<boolean name="flutter.oauthLoginAdjust" value="true" />
<boolean name="flutter.isAuthenticated" value="true" />
<boolean name="flutter.isSignedIn" value="true" />
<long name="flutter.openCount" value="1" />

<string name="flutter.token">
    83c8e0602cf540dcbe6eb815747d21ef
</string>
```

H1.2 – Plaintext authentication token and session flags in FlutterSharedPreferences.xml

This file exposes:

- Auth token stored in plaintext (flutter.token)
- Authentication state (flutter.isAuthenticated, flutter.isSignedIn)
- OAuth login flag (flutter.oauthLoginAdjust)
- User activity data (flutter.lastOpenDate, flutter.openCount)

Impact

Exposure of the plaintext authentication token can lead to:

- Account Takeover: An attacker who extracts the token could authenticate as the victim without credentials.
- Session Hijacking: The token can be replayed against backend APIs to impersonate the user.
- Data Breach: Access to the user's reading history, purchases, and account details.

Recommendation

- Use the Android Keystore to encrypt sensitive tokens before they are persisted.
- Never store raw authentication tokens in plain SharedPreferences.
- Use EncryptedSharedPreferences (Jetpack Security) instead of standard SharedPreferences.
- Implement token binding so that tokens are tied to the specific device.
- Enforce short token expiry combined with refresh-token rotation.

M1: FLAG_SECURE Bypass Enabling Unauthorized Screenshot and Screen Recording	
Score	6.8 (Medium)
Vector String	CVSS:4.0/AV:L/AC:L/AT:P/PR:N/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N
Target	Screen-capture protection logic in the book-reading activity (disableScreenshots / allowScreenshots)

Overview

ReadEase applies Android's FLAG_SECURE window flag while a book is being read, in order to prevent users from taking screenshots or screen recordings of premium content. Because this protection is enforced entirely client-side, through a simple bitmask flag (8192) on the activity window, it can be disabled at runtime using dynamic instrumentation tools such as Frida, without modifying or repackaging the APK. This finding relates to MASVS-RESILIENCE, since the application implements no anti-tampering or anti-instrumentation controls to detect or block this kind of runtime manipulation.

Details

Reviewing the decompiled source, we located the logic responsible for enabling and disabling screenshot protection:

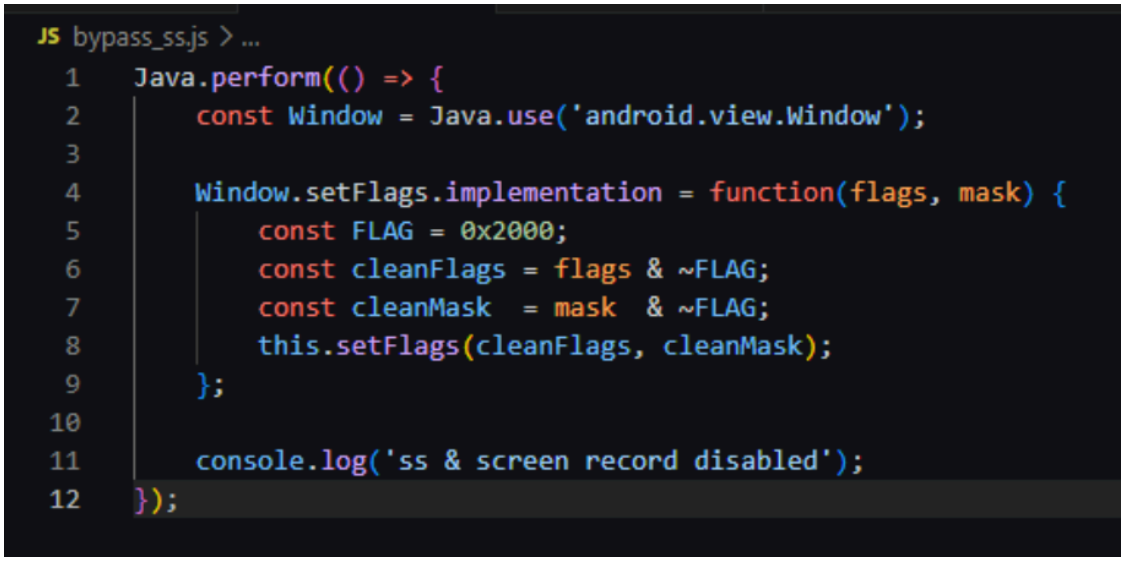
```
private final void disableScreenshots() {
    getWindow().setFlags(8192, 8192);
}

private final void allowScreenshots() {
    getWindow().clearFlags(8192);
}
```

M1.2 – Frida hook clearing the FLAG_SECURE bit at runtime

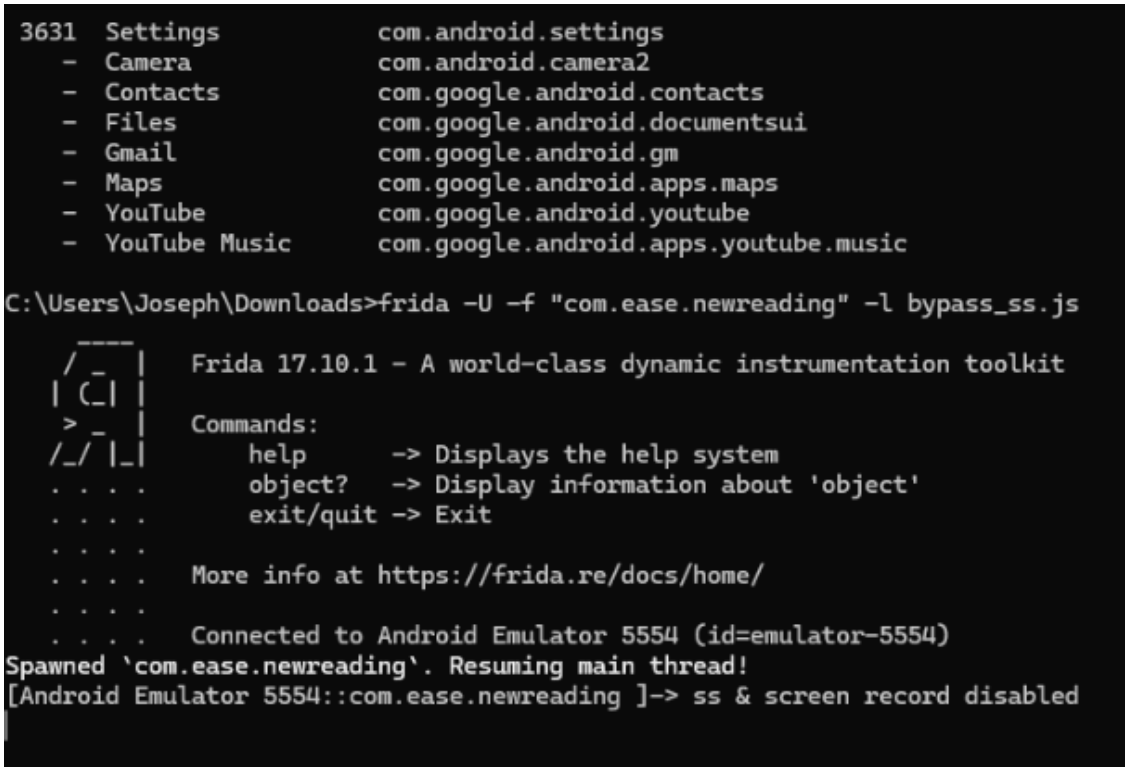
Under normal conditions, attempting to capture a screenshot inside the app is blocked by Android with a "Disabled by your admin" message, confirming

the flag is active. We then wrote a Frida script that hooks android.view.Window.setFlags and strips the FLAG_SECURE bit (0x2000 / 8192) from any flags or mask the application attempts to set, effectively forcing the protection to 0 at runtime:



M1.2 – Frida hook clearing the FLAG_SECURE bit at runtime

After injecting this script into the running application (frida -U -f "com.ease.newreading" -l bypass_ss.js), the hook confirmed the flag had been neutralized, and we were able to successfully capture a screenshot of premium, in-progress book content that would normally be protected:



M1.3 – Frida session confirming the bypass and a successful screenshot capture

Impact

- Premium content displayed on screen can be recorded or captured.
- Screen recording can occur without the developer's or content owner's knowledge.
- Premium / paid user content can be leaked or pirated.

Recommendation

- Do not rely on FLAG_SECURE as the sole protection mechanism.
- Implement server-side session and content-access verification.
- Add root-detection and instrumentation-detection (anti-Frida) checks.
- Apply additional content-level encryption for premium chapters.

M2: Unencrypted Transmission of Advertising ID (ADID)	
Score	5.3 (Medium)
Vector String	CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:P/VC:L/VI:N/VA:N/SC:L/SI:N/SA:N
Target	Advertising ID retrieval logic (getGoogleADID) and its associated network transmission

Overview

ReadEase retrieves the device's Google Advertising ID (ADID) and transmits it to the application's ad server to drive ad personalization. We found that this identifier, along with the device's Limit Ad Tracking status, is sent without being encrypted at the application layer, meaning the value is exposed in cleartext as it travels between the device and Google Play Services / the backend ad server.

Details

Within the decompiled source, we identified the method responsible for retrieving the ADID:

```
/* JADX INFO: Access modifiers changed from: private */
public static final Unit getGoogleADID$lambda$0(MainActivity mainActivity, final Function1 function1) {
    try {
        final String id2 = AdvertisingIdClient.getAdvertisingIdInfo(mainActivity).getId();
        if (id2 == null) {
            id2 = "";
        }
```

M2.1 – ADID retrieval logic in MainActivity

We then wrote a Frida hook targeting com.google.android.gms.ads.identifier.AdvertisingIdClient to intercept the ADID and the device's ad-tracking preference at the point of retrieval:

```
JS hook_adid.js > ...
1   Java.perform(() => {
2       const AdvClient = Java.use(
3           'com.google.android.gms.ads.identifier.AdvertisingIdClient'
4       );
5
6       AdvClient.getAdvertisingIdInfo.overload(
7           'android.content.Context'
8       ).implementation = function(ctx) {
9           const info = this.getAdvertisingIdInfo(ctx);
10          console.log('ADID: ' + info.getId());
11          console.log('Limit tracking: ' + info.isLimitAdTrackingEnabled());
12          return info;
13      };
14  });
```

M2.2 – Frida hook intercepting the ADID and Limit Ad Tracking flag (TRUE = not trackable, FALSE = trackable)

Injecting this script via Frida while using the application confirmed the ADID was retrievable in cleartext and that the device's Limit Ad Tracking setting returned false, meaning the identifier remains trackable:

```
Spawned `com.ease.newreading`. Resuming main thread!
[Android Emulator 5554::com.ease.newreading ]-> ss & screen record disabled
[Android Emulator 5554::com.ease.newreading ]-> exit

Thank you for using Frida!

C:\Users\Joseph\Downloads>frida -U -f "com.ease.newreading" -l hook_adid.js
     ____
    / _  |   Frida 17.10.1 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Android Emulator 5554 (id=emulator-5554)
Spawned `com.ease.newreading`. Resuming main thread!
[Android Emulator 5554::com.ease.newreading ]-> ADID: 17596752-b200-41f2-8f54-e35d05e62ef3
Limit tracking: false
ADID: 17596752-b200-41f2-8f54-e35d05e62ef3
Limit tracking: false
ADID: 17596752-b200-41f2-8f54-e35d05e62ef3
Limit tracking: false
ADID: 17596752-b200-41f2-8f54-e35d05e62ef3
Limit tracking: false
ADID: 17596752-b200-41f2-8f54-e35d05e62ef3
Limit tracking: false
```

M2.3 – Intercepted ADID and tracking status

We also observed that performing this interception appeared to interfere with the app's ability to load ads, the most likely explanation being that the

hook is detected by, or otherwise interferes with, the underlying ad SDK's own integrity checks.

Impact

- User privacy is not assured, as the device identifier is exposed in cleartext.
- Data tracking can occur without explicit user consent.
- Potential violation of data-protection regulations such as the GDPR and Indonesia's Electronic Information and Transactions Law (UU ITE).
- Possible downstream user profiling by parties able to observe the unencrypted traffic.

Recommendation

- Display a clear notification to the user before retrieving the ADID.
- Encrypt the ADID before it is transmitted to the server.
- Implement and clearly surface a user privacy policy covering ad-identifier usage.