Additional apps Results - SOEN 321 Android Apps Analysis

Adam Al Assil 40172296
Osama Kitaz 40187313
Kristofe Kessler 40206453
Abd Alwahed Haj Omar 40246177
Zaid Rasheed 40163516
Sami Aburahma 40202807
Mohammed Mousa 40206055

November 2024

1 Results Continued

1.1 OurCalling (v23) Results

The security analysis for OurCalling (v23) identifies several vulnerabilities and areas of concern:

The app is vulnerable to the Janus vulnerability, as it supports the v1 signature scheme, which exposes devices running Android 5.0-8.0 to potential exploitation. While the app is also signed with v2 and v3 schemes, relying solely on v2 or v3 is recommended to enhance security. Additionally, the app can be installed on devices running Android 4.4 (`API 19`), which are outdated and no longer receive security updates. Increasing the minimum supported SDK to at least `API 29` (Android 10) is advised to mitigate risks associated with unpatched devices.

A high-risk issue is that application data can be backed up, as `android:allowBackup="true"` is set in the manifest. This allows unauthorized users with access to the device and `ADB` (Android Debug Bridge) enabled to copy app data, potentially exposing sensitive information. Disabling this flag or implementing custom backup rules is recommended. The app also contains an exported service (`com.ourcalling.homeless.DelegationService`) with an intent filter, making it accessible to other applications. This poses a security risk as malicious apps could exploit this service. Restricting access by setting `android:exported="false"` or securing it with permissions is advised.

Another concern is that sensitive information is logged, as identified in files like `b/a/c/d.java`. Logging such data can expose private user information and should be avoided in production environments. Although no hardcoded API keys or secrets were explicitly identified, the app should be reviewed for any sensitive hardcoded values to prevent potential reverse engineering exploits.

The app implements SSL certificate pinning, which provides protection against man-in-the-middle (`MITM`) attacks. This is a positive measure that enhances secure communication over networks. However, no evidence of secure random number generation or strong cryptographic practices was noted, and a comprehensive review of the app's cryptographic implementations is suggested.

The app includes no identifiable trackers or malware-prone permissions, which aligns with

its privacy-focused purpose. However, with an overall security score of 54/100 (Medium Risk), addressing these vulnerabilities, including limiting backup capabilities, securing exported components, and enforcing modern SDK requirements, is critical to ensure better security for users and their data.

1.2 12 Steps: Addiction Recovery (v1.1.7)

The security analysis of 12 Steps: Addiction Recovery (v1.1.7) highlights several critical vulnerabilities and risks:

The app permits cleartext traffic with `android:usesCleartextTraffic="true"`, which allows unencrypted data transmission over HTTP, exposing sensitive information to potential interception and tampering. Enforcing HTTPS is essential to safeguard communications. Another high-risk issue is that the app can be installed on older Android versions (minimum `SDK 24`, Android 7.0), which are vulnerable due to lack of security updates. Increasing the minimum supported `SDK` to at least `API 29` (Android 10) is recommended to reduce exposure to unpatched vulnerabilities.

Sensitive information, including API keys, was found hardcoded in the application. Examples include:

```
"google_api_key": "AIzaSyArkUAumo_Mr8IA1RZib1H141ZL2viocB0",
"google_crash_reporting_api_key": "AIzaSyArkUAumo_Mr8IA1RZib1H141ZL2viocB0"
```

Hardcoding sensitive keys increases the app's vulnerability to reverse engineering and should be mitigated by storing such data securely using environment variables or a secure vault.

The app also demonstrates weak cryptographic practices, such as using the CBC encryption mode with PKCS5/PKCS7 padding, which is vulnerable to padding oracle attacks. This cryptographic configuration should be updated with more secure algorithms and proper integrity checks. Additionally, the use of `SHA-1` hashing is considered insecure and should be replaced with stronger algorithms like `SHA-256`.

Exported components, including activities and services, are not adequately secured. For example, `com.aboutyou.dart_packages.sign_in_with_apple.SignInWithAppleCallback` is marked as `android:exported=true` without proper access restrictions, potentially exposing it to exploitation by malicious applications. Securing these components with permissions or setting `android:exported="false"` is strongly recommended.

The app writes to external storage, leaving sensitive data accessible to other applications, and logs sensitive information in various locations. These practices violate data security principles and should be addressed by encrypting stored data and disabling sensitive logging in production environments.

The app includes two trackers: `AppsFlyer` for analytics and `Google Crashlytics` for crash reporting.

With an overall security score of 47/100 (Medium Risk), addressing these vulnerabili-

ties—particularly cleartext traffic, insecure cryptographic practices, and exported components—will significantly improve the app's security and protect user data.

1.3 Jooay (v2.4.1) Results

The security analysis of Jooay (v2.4.1) presents key vulnerabilities:

The app permits cleartext traffic (`android:usesCleartextTraffic="true"`), allowing unencrypted HTTP communication, exposing user data to interception. Enforcing HTTPS is crucial. It also supports installation on older devices (min `SDK 21`, Android 5.0), increasing vulnerability risks. Raising the minimum `SDK` to `API 29` (Android 10) is recommended. Additionally, the app is vulnerable to the Janus vulnerability due to its v1 signature scheme. It is recommended to disable v1 for security support.

The app contains hard coded sensitive information, such as these API keys:

`"firebase_database_url": "https://zeta-axiom-119505.firebaseio.com"`,
`"google_api_key": "AIzaSyDSiwdZHW6OHV9SF83WzTKQfsNFnM8mLao"`.

These values should be handled using a secure storage to avoid reverse engineering. The app also logs sensitive data, which should be disabled in production. Weak cryptography practices, including but not limited to the usage of `MD5` and `SHA-1` along with insecure random number generators weaken the security of the app. More powerful cryptographic algorithms should be used, such as `SHA-256`.

Exported components, such as `com.parse.fcm.ParseFirebaseMessagingService`, are can be accessed without restrictions, posing a risk of unauthorized access. These should be secured with explicit permissions or `android:exported="false"`.

Trackers present in the app include `Bolts Analytics`, `Facebook Login`, `Google Crashlytics`, and `Google Firebase Analytics` (*Exodus Privacy*).

The security score is 55/100 (Medium Risk). Fixing these vulnerabilities—especially cleartext traffic, hard coded secrets, and weak cryptographic implementations—is essential to protecting user data.

1.4 WEconnect (v4.8.17) Results

The security analysis of WEconnect (v4.8.17) highlights several vulnerabilities and areas of concern:

The app is vulnerable to the Janus vulnerability, as it supports the v1 signature scheme. Although it also uses v2 and v3 signatures, reliance on the v1 scheme poses risks for devices running Android 5.0-8.0. Raising the minimum supported SDK to at least `API 29` (Android 10) is recommended to mitigate issues stemming from outdated devices and vulnerabilities. Additionally, the app allows cleartext traffic, with `android:usesCleartextTraffic="true"`, which exposes network communications to potential interception. To secure communications, enforcing HTTPS across all connections is strongly advised.

The app contains hardcoded sensitive information, such as API keys and credentials. Examples include:

```
"google_api_key": "AIzaSyAT5ndNDjgnZY_szh1iD1Ec39rGzfGMUkc"
"CodePushDeploymentKey": "fLIuMqSfVGuk4uGlmYQpYo6QcriTy6NCNTVFa"
```

Such hardcoded values can be extracted through reverse engineering, compromising app security. These keys should be removed and securely managed via environment variables or a secure vault.

The app also performs insecure read/write operations to external storage, leaving data accessible to other applications. This vulnerability is evident in components such as `com/learnium/RNDeviceInfo/RNDeviceModule.java`. Sensitive data should be encrypted and stored within private app-specific storage to prevent unauthorized access. Furthermore, the app logs sensitive data, potentially exposing it during debugging or runtime. Developers should ensure that sensitive information is not logged in production.

Despite these risks, the app employs SSL certificate pinning, providing some protection against man-in-the-middle (`MITM`) attacks. However, it uses the CBC encryption mode with PKCS5/PKCS7 padding, which is vulnerable to padding oracle attacks. Secure encryption algorithms should replace this configuration to enhance cryptographic strength. Moreover, the app utilizes an insecure random number generator, which weakens the reliability of its cryptographic functions. Implementing secure randomness generation techniques is essential.

The app includes three trackers: `Google Crashlytics`, `Google Firebase Analytics`, and `Sentry`.

With an overall security score of 51/100 (Medium Risk), addressing these vulnerabilities—particularly cleartext traffic, hardcoded secrets, and weak cryptographic implementations—is critical to improving the app's security and protecting user data effectively.

1.5 Nomo - Sobriety Clocks (v5.1.141) Results

The security analysis for Nomo - Sobriety Clocks (v5.1.141) identifies several significant issues and areas of concern:

The app is vulnerable to the Janus vulnerability, as it only uses the v1 signature scheme. This poses a significant security risk for devices running Android 5.0-8.0, where the vulnerability is exploitable. To mitigate this, the app should adopt v2 or v3 signing schemes. Furthermore, the app supports a minimum SDK version of 9 (Android 2.3), allowing installation on outdated devices with known vulnerabilities. Raising the minimum SDK version to at least 29 (Android 10) is recommended for improved security.

A high-risk issue is the insecure WebView implementation, where the app ignores SSL certificate errors and accepts any certificate, making it vulnerable to man-in-the-middle (MITM) attacks. Code in `com.adobe.air.AndroidWebView.java` shows that SSL errors are bypassed, leaving sensitive communications exposed. To fix this, strict SSL validation should be enforced. Additionally, the app performs insecure random number generation, compromising cryptographic

operations, and uses CBC encryption mode with PKCS5/PKCS7 padding, which is vulnerable to padding oracle attacks. Secure encryption algorithms and proper integrity checks should be implemented.

The app also logs sensitive information in multiple files, including `com.adobe.air.utils.AIRLogger.java`, potentially exposing private data. This logging should be disabled in production. Hardcoded sensitive information, such as API keys and secrets, was found in the code, including: `"password": "Password"` and `"use_fingerprint_to_authenticate_key": "use_fingerprint_to_authenticate_key"` Such hardcoded secrets make the app vulnerable to reverse engineering and unauthorized access. These should be removed and replaced with secure runtime configurations.

The app has access to external storage, where it reads and writes data without encryption or restrictions, making it accessible to other apps. Files such as `com.adobe.air.CameraUI.java` and `com.distriqt.extension.nativewebview.utils.ContentProviderUtils.java` handle this insecurely. It is recommended to store sensitive data in app-specific internal storage and encrypt it for added protection.

The app uses multiple dangerous permissions, such as `READ_PHONE_STATE`, `GET_ACCOUNTS`, and `WRITE_EXTERNAL_STORAGE`, which could be exploited for unauthorized access to user data. Despite these issues, the app implements SSL certificate pinning to prevent basic MITM attacks, which is a positive aspect.

The analysis also detected two trackers: `Google AdMob` for advertisements (details) and `Google Firebase Analytics` for analytics (details).

With an overall security score of 35/100 (High Risk), the app requires significant improvements to mitigate the identified vulnerabilities and protect user data effectively.

1.6 Sober Grid (v3.8.0) Results

The security analysis for Sober Grid (v3.8.0) highlights multiple vulnerabilities and areas of concern:

The app is vulnerable to the Janus vulnerability, as it uses the v1 signature scheme. Devices running Android 5.0-8.0 are particularly at risk. To mitigate this, adopting v2 or v3 signing schemes is recommended. Furthermore, the app supports a minimum SDK version of 21 (Android 5.0), which allows installation on older devices with unpatched security vulnerabilities. Raising the minimum supported SDK version to at least API 29 (Android 10) is advised for improved security.

A critical issue is the insecure WebView implementation, where user-controlled code execution in WebView poses a major security risk. Additionally, the app lacks SSL certificate validation, making it vulnerable to man-in-the-middle (MITM) attacks. The use of CBC encryption mode with PKCS5/PKCS7 padding makes it susceptible to padding oracle attacks. It also employs an insecure random number generator, which compromises the cryptographic operations' strength. Sensitive data is stored insecurely in external storage, allowing unauthorized access by other apps. The following components highlight these vulnerabilities:

External Storage Access: Files such as `com.sobergrid.utils.FilePath.java` and

com.sobergrid.utils.photo.CameraPhotoCaptureImpl.java show insecure handling of external storage.

Hardcoded Sensitive Information: Several sensitive values, such as keys, credentials, and tokens, were detected in the code. For example: "apiKey": "12345-abcde" "password": "password123"
The app also copies data to the clipboard, exposing sensitive information to potential access by other apps. Sensitive information is logged in various components, including com.sobergrid.utils.LocalyticsTracker.java. Recommendations include removing sensitive information from logs, encrypting data at rest, and securing the clipboard.

Several exported activities, such as com.sobergrid.view.activity.login.LoginActivity and com.sobergrid.view.activity.main.MainActivity, are not properly protected, leaving them accessible to other applications. These should be secured with explicit permissions or set as android:exported="false".

Despite these issues, the app employs SSL certificate pinning, providing some protection against basic MITM attacks. However, the app relies on weak hashing algorithms such as SHA-1, which are known to be insecure and should be replaced with stronger algorithms like SHA-256.

The app contains multiple trackers, including Google Firebase Analytics and Facebook Analytics, which collect user data.

With an overall security score of 49/100 (Medium Risk), it is critical to address these vulnerabilities to enhance the app's security and protect user data effectively.

1.7 TinyBit - Disability Care (v3.8) Results

The security analysis of TinyBit - Disability Care (v3.8) reveals several vulnerabilities and risks:

The app is vulnerable to the Janus vulnerability, as it supports the v1 signature scheme. While it also uses v2 and v3 schemes, relying on v1 increases risks for devices running Android 5.0-8.0. Increasing the minimum SDK to at least API 29 (Android 10) is recommended for security updates and protection against unpatched vulnerabilities. Another critical issue is that the app allows cleartext traffic, with android:usesCleartextTraffic="true", exposing communications to interception and tampering. Enforcing HTTPS is essential to protect user data.

The app contains hardcoded sensitive information, including API keys and secrets. Examples from the code include:

"facebook_client_token": "8c89091319df0644f750e8d2a1f119a9",
"google_api_key": "AIzaSyAo4m0WijIaWp5SsaE9MQeVS3cXV5-WmBs",
"google_crash_reporting_api_key": "AIzaSyAo4m0WijIaWp5SsaE9MQeVS3cXV5-WmBs".

Such hardcoded values are highly susceptible to reverse engineering and should be securely managed through environment variables or secure vaults.

The app also performs insecure read/write operations to external storage, mak-

ing sensitive data accessible to other applications. This is evident in files like `com/mr/flutter/plugin/filepicker/c.java` and `ac/j.java`. Sensitive data should be encrypted and stored in private, app-specific storage. Additionally, the app logs sensitive information in files such as `g5/e0.java` and `h5/b0.java`. Developers should ensure logging does not expose private user data, especially in production.

The app employs weak cryptographic practices, using `SHA-1` and `MD5` hashing algorithms, which are known to be vulnerable to collisions. These should be replaced with stronger algorithms like `SHA-256`. The app also uses the CBC encryption mode with PKCS5/PKCS7 padding, which is vulnerable to padding oracle attacks. Implementing secure encryption algorithms with proper integrity checks is strongly advised.

Several components, such as the `com.cloudblue.tinybit.LocationService` and `AlarmService`, are exported without proper restrictions, making them accessible to other applications. Securing these components with explicit permissions or setting `android:exported="false"` is recommended. Despite these concerns, the app implements SSL certificate pinning, which helps prevent man-in-the-middle (`MITM`) attacks on secure communication channels.

The app includes four trackers: `Facebook Login`, `Facebook Share`, `Google Crashlytics`, and `Google Firebase Analytics`.

With an overall security score of 52/100 (Medium Risk), addressing these vulnerabilities—particularly cleartext traffic, hardcoded secrets, weak cryptographic implementations, and insecure storage practices—is critical to improving the app's security and protecting user data effectively.