Information Security Research Project

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

March 8, 2025

Contents

1 Abstract

This report analyzes the security and privacy risks of 14 Android apps serving vulnerable populations, such as individuals with disabilities and those struggling with addiction. The study identifies critical vulnerabilities, including weak encryption, cleartext traffic, hardcoded sensitive information, and insecure app components. These issues expose users to risks like data interception and unauthorized access. Recommendations include enforcing HTTPS, improving cryptographic practices, and securing app components. Addressing these issues is vital to protect sensitive data and enhance user safety.

2 Introduction

The increasing use of mobile applications to support vulnerable groups, such as individuals with disabilities, homeless populations, and those struggling with substance abuse, has raised significant concerns about security and privacy. These groups often face heightened risks due to their socioeconomic challenges and may have limited awareness or resources to safeguard their personal data. Although mobile applications targeting these populations can provide essential services, such as healthcare access, social support, and resource management, they can also expose users to critical vulnerabilities. As mobile technology becomes an integral part of everyday life, it is imperative to ensure that these applications are secure, respect user privacy, and do not inadvertently put individuals at greater risk.

Vulnerable groups are particularly susceptible to exploitation and harm, making it essential to address the security and privacy risks that could exacerbate their vulnerability. Studies have shown that people in these populations are more likely to fall victim to cyber threats, which can have dire real-world consequences, including financial loss, identity theft, and increased physical harm [4]. Moreover, the stigma and marginalization faced by these groups may further discourage them from reporting security breaches or seeking help.

In this report, we focus on the security and privacy analysis of 14 Android apps designed to assist people with disabilities, homeless individuals, and those with substance abuse issues. The aim is to identify and highlight potential weaknesses in these apps that could jeopardize user data or safety. Our findings reveal several concerning vulnerabilities, including issues related to data transmission, third-party trackers, lack of encryption, and unprotected app components. These issues underscore the importance of securing apps targeting vulnerable groups to protect their sensitive information and maintain their trust in digital solutions. By analyzing these apps, we aim to raise awareness about the potential risks and provide actionable recommendations to improve their security posture.

3 Findings

Our analysis reveals several vulnerabilities that affect multiple applications. Below are the instances of vulnerabilities found across multiple apps, along with numerical counts:

a. **Janus Vulnerability:** Found in 9 out of 14 apps analyzed. This vulnerability arises from the use of the v1 signature scheme, posing significant risks for devices running Android 5.0-8.0. Affected apps include Dolphin EasyReader, Sober Grid, Accessaloo, OurCalling, WEconnect, TinyBit - Disability Care, and more.

b. **Cleartext Traffic Enabled:** Found in 8 out of 14 apps analyzed. This vulnerability makes apps susceptible to data interception due to unencrypted HTTP communication. Affected apps include TinyBit - Disability Care, Relay - Quit Addiction Smarter, 12 Steps: Addiction Recovery, Dateability, Jooay, and more.

c. **Hardcoded Sensitive Information:** Found in 11 out of 14 apps analyzed. Hardcoded sensitive information such as API keys and credentials makes these apps vulnerable to reverse engineering. Affected apps include ShelterApp, Accessaloo, TinyBit - Disability Care, Relay - Quit Addiction Smarter, WEconnect, Sober Too, Dateability, Jooay, and more.

d. **Insecure WebView Implementation:** Found in 6 out of 14 apps analyzed. Insecure WebView implementations can allow user-controlled code execution and improper SSL validation, leading to severe security risks. Affected apps include Dolphin EasyReader, iAccess Life, Dateability and more.

e. **Logging of Sensitive Data:** Found in 7 out of 14 apps analyzed. Logging sensitive information could inadvertently expose user data, particularly in production environments. Affected apps include Dolphin EasyReader, ShelterApp, iAccess Life, TinyBit - Disability Care, and more.

f. **Insecure Random Number Generation:** Found in 5 out of 14 apps analyzed. Weak cryptographic practices, such as insecure random number generation, compromise the strength of cryptographic operations. Affected apps include Nomo - Sobriety Clocks, Sober Grid, Accessaloo, Relay - Quit Addiction Smarter, and more.

g. **Dangerous Permissions:** Found in 8 out of 14 apps analyzed. Excessive or dangerous permissions could be exploited if not adequately controlled. Affected apps include Sober Too, TinyBit - Disability Care, iAccess Life, WEconnect, and more.

h. **Insecure Storage of Data in External Storage:** Found in 9 out of 14 apps analyzed. Storing sensitive data in external storage without encryption leaves it accessible to other applications, violating privacy standards. Affected apps include Dolphin EasyReader, ShelterApp, Accessaloo, TinyBit - Disability Care, and more.

These findings highlight common vulnerabilities across multiple apps, indicating a significant need for better security practices, such as enforcing HTTPS, removing hard coded secrets, and adopting more robust encryption methods. Addressing these vulnerabilities is critical to protect the sensitive information of vulnerable users and improve the overall security of mobile applications.

Vulnerability	**Number of Affected Apps**
Janus Vulnerability	9/14
Cleartext Traffic Enabled	8/14
Hardcoded Sensitive Information	11/14
Insecure WebView Implementation	6/14
Logging of Sensitive Data	7/14
Insecure Random Number Generation	5/14
Dangerous Permissions	8/14
Insecure Storage of Data in External Storage	9/14

Table 1: Summary of Vulnerabilities Found in Analyzed Applications

4 Analysis Methodology

This section describes the selection procedure for Android test applications as well as the static and dynamic analysis techniques used in the study. The analysis focuses on the identification of key vulnerabilities and the potential implications for users. The study also extends to any associated websites where applicable, ensuring a comprehensive review of all platforms that interact with these vulnerable populations.

4.1 App Selection

The researchers conducted a search on the Google Play Store using keywords specifically tailored to vulnerable groups, including terms such as *disability*, *homeless*, *accessibility*, and *mental health*. This search identified 28 Android applications. Subsequently, these apps were filtered to retain only those designed for vulnerable populations, excluding any that required sensitive financial or identity-related information. Each application was manually assessed to ensure alignment with the study's criteria. The final dataset comprised 14 applications, which collectively amassed over one million downloads.

4.2 Dynamic Analysis of Traffic Flow

Dynamic analysis was conducted to simulate real-world app usage and evaluate app behavior. This involved setting up test environments, creating user accounts, and emulating user interactions with the apps. Traffic from the apps was collected for analysis using tools such as *Frida*, *Burp Suite*, and *adb*. Burp Suite was utilized to examine network traffic and identify security vulnerabilities, including weak session management and insecure data transmission. Frida facilitated bypassing SSL pinning. The collected data was used to identify privacy issues such as PII and PHI leakage, plain-text traffic, weak authentication mechanisms, and improper access controls. The analysis was conducted using various android emulators through Android Studio.

4.3 Static Analysis: Library, App Code, and Privacy Policies

Static analysis served as a complement to dynamic testing by thoroughly examining app code and configurations to uncover privacy and security vulnerabilities. Tools such as *MobSF*, an all-in-one mobile application testing framework, were employed to analyze app binaries for issues including improper SQLite database implementations, SSL misconfigurations, and flawed WebView setups. Additionally, *Apktool* was used to decompile and recompile APK files, allowing for a detailed review of app resources and source code. Jadx was also used for in-depth analysis of decompiled apk files. Privacy policies of the analyzed applications were manually evaluated to ensure compliance with privacy standards and to identify potential inconsistencies or oversights. This comprehensive approach aimed to detect and address privacy and security risks across the selected applications.

5 Results

This section will go into details for 7 application we tested into more details individually. The detailed analysis of the remaining 7 apps can be found through **Our GitHub Link**. Below, you can find a high level table summarizing our findings:

Vulnerability	Dolphin EasyReader	Sober Too	ShelterApp	Nomo - Sobriety Clocks	Sober Grid	Accessaloo	OurCalling	WEconnect	TinyBit - Disability Care	Relay - Quit Addiction Smarter	iAccess Life	12 Steps	Dateability	Jooay
Janus Vulnerability	X	X	X	X	X	X	X	X						
Cleartext Traffic Enabled	X					X			X	X	X	X	X	X
Hardcoded Sensitive Information		X	X	X	X	X		X	X	X	X		X	X
Insecure WebView Implementation	X			X	X					X	X		X	
Logging of Sensitive Data	X	X	X	X		X			X		X			
Insecure Random Number Generation		X		X	X	X				X				
Dangerous Permissions		X	X	X		X			X	X	X			
Insecure Storage of Data in External Storage	X		X	X	X	X		X	X	X	X			

Table 2: Breakdown of Vulnerabilities found in Analyzed Application

5.1 Dolphin EasyReader v11.04 Results

The analysis reveals several significant security vulnerabilities in the Dolphin EasyReader v11.04 app: Cleartext traffic is enabled, allowing the use of HTTP, which exposes sensitive data to attackers through eavesdropping or modification. It is recommended to enforce HTTPS by setting `android:usesCleartextTraffic="false"`. The app is also vulnerable to the Janus vulnerability, as it is signed only with the v1 signature scheme, posing risks for Android 5.0-8.0 devices. This can be mitigated by adopting v2 or v3 signing schemes. Additionally, remote WebView debugging is enabled, permitting potential injection of malicious scripts into WebViews, which should be disabled in production builds. The `OpenFileContentProvider` is unprotected and exported, leaving it accessible to other apps. Restricting access with permissions or setting `android:exported="false"` is advised. Several activities, such as `ImportContentActivity`, `DeepLinkActivity`, and `RedirectUriReceiverActivity`, are similarly unprotected and need explicit permissions or restricted export settings to prevent exploitation.

Other issues of medium severity include the app's read/write permissions for external storage, which may lead to data theft or modification; the use of hardcoded sensitive information, such as keys or passwords; and clipboard data exposure, as the app copies data to the clipboard, which

can be accessed by other applications. Recommendations include adopting scoped storage APIs, removing hardcoded sensitive data, and avoiding copying sensitive data to the clipboard. The insecure WebView implementation allows the execution of user-controlled code, posing a critical risk. Restricting JavaScript execution and validating inputs are recommended to address this.

Additionally, the app logs sensitive data, which could inadvertently expose user information. Developers should ensure no sensitive data is logged in production. Certain binaries lack stack canaries and RELRO, making them vulnerable to memory corruption exploits. Compiling with `-fstack-protector-all` and enabling full RELRO is advised to mitigate these risks. The app includes trackers such as Google Crashlytics for crash reporting and Google Firebase Analytics for analytics tracking. Overall, the app's security score is 49/100 on MobSF, indicating a medium risk, and addressing these issues is critical to improving security.

5.2 Sober Too (v1.4.3) Results

The security analysis for the app Sober Too (v1.4.3) highlights several significant issues:The app is vulnerable to the Janus vulnerability because it supports the v1 signature scheme, which can be exploited on devices running Android 5.0-8.0. This risk can be mitigated by ensuring the app is signed using v2 or v3 signature schemes. Furthermore, the app does not enforce a minimum Android version of API 29 (Android 10), meaning it can be installed on outdated, unpatched devices, exposing it to numerous security vulnerabilities. The app also employs CBC encryption mode with PKCS5/PKCS7 padding, which is susceptible to padding oracle attacks. To address this, a more secure encryption algorithm should be implemented. Additionally, the app contains hardcoded sensitive information, such as keys and credentials, making it vulnerable to reverse engineering. Removing these hardcoded secrets and employing secure key management practices is crucial.

The app requests dangerous permissions, such as access to the camera, microphone, external storage, and notification posting, which could be exploited for data theft or misuse if not properly controlled. Another critical issue is the app's use of insecure random number generation, potentially weakening cryptographic operations. It also copies sensitive data to the clipboard, making it accessible to other applications, and logs potentially sensitive information, which could be exploited by attackers.

Despite these vulnerabilities, the app does implement SSL certificate pinning, which protects against man-in-the-middle (MITM) attacks on secure communication channels. However, the app uses outdated cryptographic algorithms such as SHA-1 and MD5, known for their weaknesses, and these should be replaced with stronger hashing algorithms like SHA-256.

Finally, the app includes one tracker, Google Crashlytics, for crash reporting (details). With an overall security score of 55/100 (Medium Risk), addressing these vulnerabilities is essential to enhance the app's security posture.

5.3 ShelterApp (v2.0.11) Results

The security analysis of ShelterApp (v2.0.11) reveals several significant vulnerabilities:

The app is vulnerable to the Janus vulnerability, as it supports the v1 signature scheme, which

puts devices running Android 5.0-8.0 at risk. To address this, v2 or v3 signing schemes should be enforced. Furthermore, the app can be installed on devices with outdated Android versions (min SDK 21, Android 5.0), exposing it to numerous unpatched vulnerabilities. Increasing the minimum supported version to API 29 (Android 10) is recommended. Several components, including activities (FacebookActivity, CustomTabActivity), services (RNFirebaseMessagingService), and receivers (RNDeviceReceiver), are exported without proper protection (android:exported=true), leaving them open to exploitation by other applications. These components should either be secured with appropriate permissions or have android:exported="false" explicitly set.

Other medium-severity issues include the use of raw SQL queries, which can lead to SQL injection if user input is not properly sanitized, and the use of insecure WebView implementation, which allows execution of user-controlled code, posing a critical security risk. The app also logs sensitive information, increasing the risk of exposing private user data. The app contains hardcoded sensitive information, including API keys and tokens, making it highly vulnerable to reverse engineering. Examples of such sensitive data found in the code include:

`"GOOGLE_MAPS_APIKEY": "AIzaSyDg2iZZy3zugO2Uo-brrUcfRv6khyQSoKo",`
`"TWITTER_CONSUMER_KEY": "GkOT1u6XJjPt5eQOWsQOiBTdS",`
`"TWITTER_CONSUMER_SECRET": "z9OpBbQHCEtLEvyHP9tyZuFx03Xr1mSn8dQDfkX5l3tj1hUW1U",`
`"firebase_database_url": "https://shelterapp1573928197721.firebaseio.com".`

Hardcoding these sensitive keys exposes them to attackers using reverse engineering tools, which can compromise the security and back-end services of the application. These keys should be removed from the code and managed securely using environment variables or a dedicated API key management service.

Additionally, the app's handling of external storage further exacerbates the security risk. The code in files such as `com.imagepicker.utils.MediaUtils.java`, `com.reactnativecommunity.webview.RNCWebViewModule.java`, and `io.invertase.firebase.storage.RNFirebaseStorage.java` shows that the app performs read and write operations to external storage without implementing encryption or access control mechanisms. This leaves sensitive user data stored in external locations accessible to any other app on the device, violating standard data security protocols. To mitigate this, it is essential to adopt secure storage practices, such as encrypting files and using private app-specific storage, to prevent unauthorized access. Additionally, the app has access to external storage, where any app can read the written data, violating data security standards. Implementing secure storage mechanisms is crucial.

The app uses multiple dangerous permissions, such as camera, fine/coarse location, and external storage access, which could be exploited for data theft if not managed carefully. However, it implements SSL certificate pinning, offering protection against man-in-the-middle (MITM) attacks.

The app includes five trackers: Facebook Analytics, Facebook Login, Facebook Places, Facebook Share, and Google Firebase Analytics.

With an overall security score of 48/100 (Medium Risk), addressing these vulnerabilities is essential to improve the app's security posture and protect user data effectively.

5.4 Accessaloo (v1.2.0) Results

The security analysis for Accessaloo (v1.2.0) highlights several vulnerabilities and concerns:

The app is vulnerable to the Janus vulnerability, as it supports the v1 signature scheme, making it susceptible to exploitation on devices running Android 5.0-8.0. While it also uses v2 and v3 signatures, supporting only the v2 or v3 scheme is recommended to mitigate risks. The app can be installed on devices with a minimum SDK version of 21 (Android 5.0), allowing installation on outdated, vulnerable devices. It is recommended to raise the minimum SDK version to at least API 29 (Android 10) to enhance security.

Another high-risk issue is the use of cleartext traffic, with `android:usesCleartextTraffic="true"`, allowing unencrypted data transmission over HTTP. This makes the app vulnerable to data interception and tampering. Enforcing HTTPS is essential to address this issue. The app performs insecure read and write operations to external storage, where data is stored unencrypted and accessible to other apps. Code in components such as `com.disabledaccessibletravel.accesaloo.utils.FileUtil.java` demonstrates this insecure handling of files. Sensitive data should be stored in app-specific internal storage with proper encryption.

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

```
"firebase_database_url": "https://accessaloo-212616.firebaseio.com"
"google_api_key": "AIzaSyAKbspaiKIR6t84WFS0p9cFEdv8-BIk27E"
"password": "Password..."
```

Such hardcoded values are easily extractable via reverse engineering and should be replaced with secure runtime configurations or stored securely.

Despite these concerns, the app implements SSL certificate pinning, which helps protect against man-in-the-middle (MITM) attacks. However, it uses an insecure random number generator, as shown in components like `com.disabledaccessibletravel.accesaloo.service.LocationService`, which weakens the security of cryptographic operations. Adopting secure random number generation practices is advised.

The app includes two trackers: `Google Crashlytics` for crash reporting (details) and `Google Firebase Analytics` for analytics (details).

With an overall security score of 53/100 (Medium Risk), the app requires improvements to address these vulnerabilities, including eliminating hardcoded secrets, enforcing secure network communication, and securing data storage. These actions are critical to enhancing user data protection and overall application security.

5.5 Relay - Quit Addiction Smarter (v1.989.2) Results

The security analysis for Relay - Quit Addiction Smarter (v1.989.2) highlights multiple vulnerabilities:

The app is vulnerable to the Janus vulnerability, as it relies on v2 and v3 signature schemes but lacks v1 compatibility. This presents a risk for devices running Android 5.0-8.0. The app also supports a minimum SDK of `26` (Android 8.0), which, while reasonable, leaves room for better security by increasing the minimum SDK to at least `API 29` (Android 10). Another critical issue is the presence of cleartext traffic, with `android:usesCleartextTraffic="true"`, exposing sensitive data to potential interception. Enforcing HTTPS across all communications is strongly advised.

The app contains hardcoded sensitive information, such as API keys and tokens. Examples from the code include:

`"firebase_database_url": "https://tribe-f8009-default-rtdb.firebaseio.com"`,
`"google_api_key": "AIzaSyDY5yEdOEDiFCIrxzTHP2ftYLuZhZ2mW7Q"`,
`"facebook_client_token": "26fbbf907375fe93b2df2e5e6a837d73"`.

Such values are susceptible to reverse engineering and should be securely stored or managed through environment variables. Additionally, logging sensitive data is a recurring issue across various app components, which can expose user information. Developers should ensure sensitive data is not logged, particularly in production.

The app demonstrates weak cryptographic practices, using insecure hash algorithms like `MD5` and `SHA-1`, both of which are vulnerable to collisions. These should be replaced with stronger algorithms such as `SHA-256`. The app also employs CBC encryption with PKCS5/PKCS7 padding, which is vulnerable to padding oracle attacks, and insecure random number generators. Cryptographic practices should be reviewed and upgraded to secure standards.

Certain app components, such as:
`io.flutter.plugins.firebase.messaging.FlutterFirebaseMessagingReceiver`, are exported without adequate protection, making them accessible to other apps. These components should be restricted or secured with appropriate permissions. The app's WebView implementation is insecure, allowing execution of user-controlled code, and remote debugging is enabled, both of which expose the app to critical security risks. Disabling WebView debugging and sanitizing inputs are critical steps to mitigate these vulnerabilities.

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

With an overall security score of 46/100 (Medium Risk), addressing these vulnerabilities—particularly cleartext traffic, hardcoded secrets, and insecure WebView implementation—is essential to enhance the app's security and protect user data effectively.

5.6 iAccess Life (v3.0) Results

The security analysis for iAccess Life (v3.0) identifies several vulnerabilities and issues:

The app is vulnerable to the Janus vulnerability, as it uses the v1 signature scheme. This presents a risk for devices running Android 5.0-8.0, as these versions are susceptible to exploitation. Raising the minimum supported SDK to at least `API 29` (Android 10) is recommended for better protection. Furthermore, the app permits cleartext traffic, with `android:usesCleartextTraffic="true"`, allowing unencrypted data transmission that exposes sensitive information to interception. Enforcing HTTPS for all communications is essential.

A significant risk is the presence of hardcoded sensitive information in the application code. Examples include:
`"google_api_key": "AIzaSyBY5cU525pWqrA9ymWYKX57JLPkeNjFfQw"`,
`"branchio_test_key": "key_test_gmNiVADHl2V1SwzBc1Y2CakctEgXGVrn"`,
`"firebase_database_url": "https://iaccess-innovations-inc.firebaseio.com"`

Hardcoded secrets are highly vulnerable to reverse engineering and should be securely stored using environment variables or secure vaults.

The app also performs insecure read/write operations to external storage, as seen in files like `com/iaccess/utils/ImagePickerController.java` and `com/mikelau/croperino/CroperinoFileUtil.java`. Sensitive data stored in external storage can be accessed by other apps, violating data privacy. Encrypting data and using private app-specific storage is advised. Additionally, the app logs sensitive data in multiple locations, including `com/iaccess/utils/GetCurrentLocation.java`, which can expose user data during runtime. Logging sensitive information should be strictly avoided in production.

The app employs weak cryptographic practices, using outdated algorithms like `MD5` and `SHA-1`, which are susceptible to collisions and should be replaced with stronger options like `SHA-256`. It also uses CBC encryption with PKCS5/PKCS7 padding, which is vulnerable to padding oracle attacks. Furthermore, the app utilizes an insecure random number generator, which compromises the security of its cryptographic functions. These cryptographic weaknesses should be addressed by adopting secure algorithms and proper implementation practices.

Several components, such as `com.facebook.CustomTabActivity` and `com.google.android.gms.auth.api.signin.RevocationBoundService`, are exported without adequate restrictions, leaving them accessible to other applications. These components should be secured with appropriate permissions or by setting `android:exported="false"`. Additionally, the app includes an insecure WebView implementation, where user-controlled code execution is possible. To mitigate this, developers should sanitize inputs and disable remote debugging for WebViews.

The app includes seven trackers: `Google Crashlytics`, `Google Firebase Analytics`, `Google AdMob`, `Branch Analytics`, `Facebook Analytics`, `Facebook Login`, and `Facebook Share`. These trackers potentially collect user data for various purposes.

With an overall security score of 49/100 (Medium Risk), addressing these vulnerabili-

ties—particularly cleartext traffic, hardcoded secrets, and insecure WebView implementation—is critical to improving the app's security and ensuring better protection of user data.

5.7 Dateability (v59) Results

The security analysis of Dateability (v59) has several issues and areas that need improvement:

The app is vulnerable to the Janus vulnerability because it supports the v1 signature scheme. It also uses v2 and v3 schemes, but keeping v1 support can allow it to be exploited on devices running Android 5.0-8.0. The minimum supported `SDK` should be increased to at least `API 29` (Android 10) which is highly recommended for better protection against known vulnerabilities. Additionally, the app allows sending cleartext traffic, with `android:usesCleartextTraffic="true"`, enabling the transmission of unencrypted data, hence making sensitive user information susceptible to interception. Enforcing the usage of HTTPS on all communication channels is essential.

A huge issue is that hard coded sensitive information is shown, such as API keys, which can be reverse engineered. Examples include:

`"google_api_key": "AIzaSyCaVLpQnSUwuwSCFsX4LUZmPtD81XkxocQ"`,
`"google_crash_reporting_api_key": "AIzaSyCaVLpQnSUwuwSCFsX4LUZmPtD81XkxocQ"`.

These are information that should be protected through environment variables or secure methods of storing keys to avoid exposure.

Logging sensitive data should not be done because it risks the exposure of the user's private data. Logging sensitive data should be strictly avoided in production environments. Also, the cryptography used in the app is very weak. For example, `SHA-1` hashing, is used, which is vulnerable to collisions, and CBC encryption with PKCS5/PKCS7 padding, which is susceptible to padding oracle attacks. These should be replaced with stronger algorithms like `SHA-256` and secure encryption methods.

The app contains components like `com.google.firebase.iid.FirebaseInstanceIdReceiver` and `com.amazon.device.iap.ResponseReceiver` marked as `android:exported=true`, which means other apps can access them. These components should be secured with appropriate permissions or set as `android:exported="false"`. Similarly, the app reads/writes data to the external storage in an insecure manner, where any other application can access it. This is a violation of data privacy, and such sensitive data should be encrypted and stored instead in private app-specific directories.

Despite these issues, the app has set up SSL Certificate pinning that protects it against the man-in-the-middle (`MITM`) attacks; nonetheless, several improvements are required for complete assurance of network communication.

The app includes one tracker, `Google Firebase Analytics`, which collects user data for analytic purposes.

With an overall security score of 55/100 (Medium Risk), the main issues to be addressed are cleartext traffic, hard coded secrets, and weak cryptographic practices in order to improve

security and protect user data effectively.

6 Conclusion

This study has highlighted a number of security and privacy vulnerabilities in android apps that target vulnerable groups, such as the disabled, the homeless, or people with substance abuse issues. The major key findings involve weak encryption practices, transmission using cleartext data, hardcoding sensitive information, and insecure app components, which introduce high risks, including data interception, unauthorized access, and exposure of personal data. Being able to address these issues means ensuring the safety, privacy, and trust of these vulnerable users.

Developers can reduce these risks by providing better cryptographic techniques, data protection methods, and secure architectures of the apps. This way, these apps will help in protecting the users' privacy more and contribute positively toward their well-being. In addition, this paper should encourage the establishment of more strict security guidelines, regular security checks, and raising awareness about the importance of security in android apps targeting vulnerable people.

References

[1] APKPure. Apkpure: Android apk downloads, 2024. Accessed: 2024-11-29.

[2] APKTool. Apktool: Android application reverse engineering, 2024. Accessed: 2024-11-29.

[3] Frida. Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers, 2024. Accessed: 2024-11-29.

[4] Nora McDonald, Karla Badillo-Urquiola, Morgan G. Ames, Nicola Dell, Elizabeth Keneski, Manya Sleeper, and Pamela J. Wisniewski. Privacy and power: Acknowledging the importance of privacy research and design for vulnerable populations. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI EA '20, page 1–8, New York, NY, USA, 2020. Association for Computing Machinery.

[5] MobSF. Mobile security framework (mobsf), 2024. Accessed: 2024-11-29.

[6] PortSwigger. Burp suite community edition, 2024. Accessed: 2024-11-29.

[7] OWASP Mobile Application Security Project. Mobile application security testing guide (mastg), 2024. Accessed: 2024-11-29.

[8] OWASP Mobile Application Security Project. Owasp checklists, 2024. Accessed: 2024-11-29.

[9] Skylot. Jadx: Command-line and gui tools for android decompilation, 2024. Accessed: 2024-11-29.