

Static analysis of AndroidManifest.xml and literal extraction - Whistle Time

Introduction:

Before delving into the details of permissions and components, it is important to understand how we obtained this `AndroidManifest.xml`. We use **apktool** To decompile the Whistle Time APK and extract all Android resources and settings:

```
apktool d WhistleTime-release.apk -o wt-apktool
```






This command generates the folder `wt-apktool/`, where the file is located `AndroidManifest.xml` which we analyze below.

1. Declared Permissions

Description:

In this section we analyze the permissions requested by the Whistle Time application, evaluating their potential risk and justification to determine their security relevance.

Findings:

- **android.permission.INTERNET**: Necessary for communication with external APIs. Standard use. 
- **android.permission.RECEIVE_BOOT_COMPLETED**: Allows the app to receive notifications after restarting the device. It can be risky if not used correctly. 
- **android.permission.FOREGROUND_SERVICE** and **FOREGROUND_SERVICE_DATA_SYNC**: Required to run foreground services and constantly synchronize data. 
- **android.permission.WAKE_LOCK**: Prevents the CPU from going to sleep during critical tasks. Fair use. 
- **android.permission.ACCESS_NETWORK_STATE**: Allows you to check the availability of the network connection. Fair use. 

- **android.permission.POST_NOTIFICATIONS** and **VIBRATE**: Necessary for notifications and alerts in the app. Fair use. ✓

Evidence:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:compileSdkVersion="35" android:compileSdkVersionCodename="15" package="com.whistletime.app" platformBuildVersionCode="35" platformBuildVersionName="1.0">
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE_DATA_SYNC"/>
    <queries>
        <intent>
            <action android:name="android.intent.action.VIEW"/>
            <data android:mimeType="application/pdf"/>
        </intent>
        <intent>
            <action android:name="android.intent.action.SEND"/>
            <data android:mimeType="application/pdf"/>
        </intent>
        <intent>
            <action android:name="android.intent.action.PROCESS_TEXT"/>
            <data android:mimeType="text/plain"/>
        </intent>
    </queries>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
    <permission android:name="com.whistletime.app.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION" android:protectionLevel="signature"/>
    <uses-permission android:name="com.whistletime.app.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION"/>
</manifest>
```

Recommendations:

- Strictly verify the need for permission **RECEIVE_BOOT_COMPLETED**. If not strictly necessary, disable it to reduce attack surface.
- Continue to periodically review requested permissions to maintain least privilege and better protect user privacy.

2. Exported components

Description:

In this section we review which activities, services and recipients are marked as **android:exported="true"**. Exported components can be invoked by other apps or by the system, so make sure that only the necessary ones are accessible.

2.1 Exported activities

Component	Exported	Justification	Potential risk
-----------	----------	---------------	----------------

MainActivity (com.whistletime.app.MainActivity)	true	App entry point (MAIN/LAUNCHER)	Low (required)
GenericIdpActivity (com.google.firebase.auth.internal.GenericIdpActivity)	true	Callbacks OAuth de Firebase	Low (part of Firebase)
RecaptchaActivity (com.google.firebase.auth.internal.RecaptchaActivity)	true	Firebase Auth Captcha	Low (part of Firebase)

Finding 2.1: All exported activities are required for the app startup flow and for integration with Firebase Auth.

2.2 Exported services

Component	Exported	Justification	Potential risk
RevocationBoundService (com.google.android.gms.auth.api.signin.RevocationBoundService)	true	Google Sign-In Revocation Service	Low (part of Google Play)
Rest of services	false	—	—

Finding 2.2: Only the Google Sign-In revocation service is exported and provided by Play Services. There are no exported app services.

```
<service android:exported="true" android:name="com.google.android.gms.auth.api.signin.RevocationBoundService"
```

2.3 Exported Receivers

Component	Exported	Justification	Potential risk
-----------	----------	---------------	----------------

<code>RebootReceiver</code> (<code>com.pravera.flutter_foreground_task.service.RebootReceiver</code>)	true	Restart foreground tasks after reboot or update	Medium (allows execution in boot)
<code>ProfileInstallReceiver</code> (<code>androidx.profileinstaller.ProfileInstallReceiver</code>)	true	Installing performance profiles	Low (AndroidX component)

Finding 2.3:

- `RebootReceiver` is exported and handles the event `BOOT_COMPLETED`. This is necessary to relaunch the foreground service on startup, but increases the attack surface if a malicious actor manages to send false intents.
- The AndroidX profile receiver is standard and poses no risk.

```
<receiver android:enabled="true" android:exported="true" android:name="com.pravera.flutter_foreground_task.service.RebootReceiver">
  <intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED"/>
    <action android:name="android.intent.action.MY_PACKAGE_REPLACED"/>
    <action android:name="android.intent.action.QUICKBOOT_POWERON"/>
  </intent-filter>
</receiver>
```

Recommendations for Exported Components

1. **Verify accepted intents** by `RebootReceiver` and filter only the necessary actions.
2. If restarting the service at boot is not required, consider disabling `RECEIVE_BOOT_COMPLETED` and/or put `android:exported="false"` in that receiver.
3. Keep it documented in your README that these exported components are intended and what flow they are for.

3. Providers y FileProviders

Description:

In this section we review the `<provider>` defined in the manifest, paying attention to `android:exported` and permissions granted. Content providers can expose internal URIs of the app, so they must be configured correctly.

3.1 Own FileProviders

Provider	Authorities	Exported	Grant URI Permissions	Potential risk
<code>FileProvider</code> (<code>androidx.core.content.FileProvider</code>) <code>com.whistletime.app.fileprovider</code>	<code>com.whistletime.app.fileprovider</code>	false	true	Low (controlled access only)
<code>ImagePickerFileProvider</code> (<code>io.flutter.plugins.imagepicker.ImagePickerFileProvider</code>) <code>com.whistletime.app.flutter.image_provider</code>	<code>com.whistletime.app.flutter.image_provider</code>	false	true	Bass (Flutter plugin)
<code>PrintFileProvider</code> (<code>net.nfet.flutter.printing.PrintFileProvider</code>) <code>com.whistletime.app.flutter.printing</code>	<code>com.whistletime.app.flutter.printing</code>	false	true	Bass (Flutter plugin)

Finding 3.1:

All FileProviders are configured with `android:exported="false"` and `grantUriPermissions="true"`, which is the recommended setting: they allow sharing only those files that you explicitly grant via `Intent.setData()` the `Intent.grantUriPermission()`.

3.2 Initialization providers and Firebase

Provider	Authorities	Exported	Direct Boot Aware	Potential risk
----------	-------------	----------	-------------------	----------------

<code>FirebaseInitProvider</code> (<code>com.google.firebase.provider.FirebaseInitProvider</code>) <code>com.whistletime.app.firebaseinitprovider</code>	<code>com.whistletime.app.firebaseinitprovider</code>	<code>false</code>	<code>true</code>	Low (initializes Firebase)
<code>InitializationProvider</code> (<code>androidx.startup.InitializationProvider</code>) <code>com.whistletime.app.androidx-startup</code>	<code>com.whistletime.app.androidx-startup</code>	<code>false</code>	<code>false</code>	Low (AndroidX Startup)

Finding 3.2:

The initialization providers (`FirebaseInitProvider`, `InitializationProvider`) are correctly not exported, with `android:exported="false"`. The first is `directBootAware="true"`, which allows Firebase to be initialized even before unlocking the device, necessary for notifications and background tasks.

```
<provider android:authorities="com.whistletime.app.firebaseinitprovider" android:directBootAware="true" android:exported="false" />
```

Recommendations for Providers

1. **Keep `exported="false"`** in all app providers to prevent unauthorized access to internal URIs.
2. Periodically review the content of the paths defined in `file_paths.xml` and in the meta-data of each `FileProvider` to ensure that they do not expose sensitive directories.

4. Network Security Config and Network Traffic

Description:

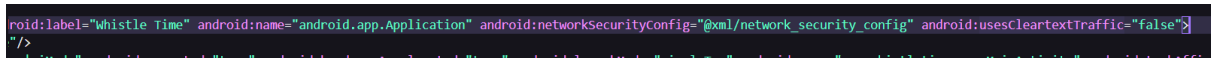
Here we evaluate the network security configuration declared in the manifest, focusing on `android:networkSecurityConfig` and traffic policy in clear (`usesCleartextTraffic`). This tells us if the app allows insecure HTTP communications or restricts traffic to encrypted channels.

4.1 usesCleartextTraffic="false"

```
<application
    ...
    android:networkSecurityConfig="@xml/network_security_config"
    android:usesCleartextTraffic="false">
    ...
</application>
```

Finding 4.1:

The property `usesCleartextTraffic="false"` is correctly established, which **prohibits all HTTP traffic in the clear** by default in Android 9+ except for exceptions that are defined in the `network_security_config`.



```
android:label="@string/whistle_time" android:name="android.app.Application" android:networkSecurityConfig="@xml/network_security_config" android:usesCleartextTraffic="false"
/>
```

4.2 network_security_config

- Point to file `res/xml/network_security_config.xml`.
- You should review that XML to verify allowed domains and certificate trust settings.

A typical minimum example would be:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain
includeSubdomains="true">api.whistletime.com</domain>
        </domain-config>
        <base-config>
            <trust-anchors>
                <certificates src="system"/>
                <!-- Cert pinning option -->
            </trust-anchors>
        </base-config>
    </network-security-config>
```

-

Finding 4.2 (pending):

It remains to be confirmed that in `network_security_config.xml` only allow traffic to your API domains and completely disable any exceptions to unencrypted HTTP. Furthermore, it is convenient to define `certificates src="system"` and, optionally, pin critical certificates.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <debug-overrides>
    <trust-anchors>
      <certificates src="user" />
    </trust-anchors>
  </debug-overrides>
  <base-config cleartextTrafficPermitted="false">
    <trust-anchors>
      <certificates src="system" />
    </trust-anchors>
  </base-config>
  <domain-config cleartextTrafficPermitted="true">
    <domain includeSubdomains="true">dev-login.miapi.com</domain>
  </domain-config>
</network-security-config>
```

Network Security Recommendations

1. **Review and reinforce** `network_security_config.xml`:
 - Make sure **there is not** `<domain-config cleartextTrafficPermitted="true">` for insecure domains.
 - Declare only the domains from the backend (eg `api.whistletime.com`).
2. **Implementar Certificate Pinning** in the most critical HTTP/HTTPS calls (for example, in Retrofit or HttpClient).
3. **Audit** that all third-party libraries do not introduce exceptions to the clear traffic policy.
4. **Prove** with the tool **Network Security Configuration Tester** from Android Studio to validate that unencrypted traffic is not allowed.

Description:

We review the content of `res/xml/network_security_config.xml` to check for exceptions to encrypted traffic and certificate trust settings.


```
<network-security-config>
  <debug-overrides>
    <trust-anchors>
      <certificates src="user" />
    </trust-anchors>
  </debug-overrides>
  <base-config cleartextTrafficPermitted="false">
    <trust-anchors>
      <certificates src="system" />
    </trust-anchors>
  </base-config>
  <domain-config cleartextTrafficPermitted="true">
    <domain
includeSubdomains="true">dev-login.miapi.com</domain>
    </domain-config>
  </network-security-config>
```

Findings

1. **debug-overrides** trusts user certificates

- In debug mode, any certificate installed on the device will be trusted.
- **Risk:** Allows MITM attacks during testing if someone installs a malicious certificate.
- *(insert screenshot of section <debug-overrides>)*

2. **Base-config** prohibits clear traffic

- `<base-config cleartextTrafficPermitted="false">` By default it ensures only HTTPS.
- *(insert screenshot of <base-config>)*

3. **HTTP exception** for development domain

- `<domain-config cleartextTrafficPermitted="true">` allows HTTP on `dev-login.miapi.com` and its subdomains.
- **Risk:** If this domain is used in production by mistake, the traffic could go in plain text.

- (insert screenshot of `<domain-config>` showing development domain)

Recommendations

1. Eliminate conditioning `debug-overrides` in release

- Ensure that `debug-overrides` is only present in development builds.
- In production, do not trust user certificates.

2. Restrict cleartext exceptions to debug builds only

- Move the `<domain-config cleartextTrafficPermitted="true">` still `network_security_config_debug.xml` separate.
- In the release config, completely remove any HTTP exceptions.

3. Verify Production API Domain

- Confirm that `dev-login.miapi.com` not be used in actual deployments.
- If an unencrypted endpoint is needed, consider a secure tunnel or HTTPS proxy.

5. Firebase Configuration (`google-services` and XML)

Description:

In this section we extract and analyze the Firebase embedded values found in `res/values/strings.xml`, required to initialize Firebase services in the app.

```
<resources>
```

```
...
```

```
<string name="gcm_defaultSenderId">1029243904025</string>
```

```

    <string
name="google_api_key">AIzaSyAKM1lXRRN5ge3NDV8StR80V1qN94BRwkE</string>

    <string
name="google_app_id">1:1029243904025:android:41a476a67ac37f43ed8a88<
/string>

    <string
name="google_crash_reporting_api_key">AIzaSyAKM1lXRRN5ge3NDV8StR80V1
qN94BRwkE</string>

    <string
name="google_storage_bucket">whistle-time-7cc21.firebaseiostorage.app<
/string>

    <string name="project_id">whistle-time-7cc21</string>

    ...

</resources>

```

```

<string name="google_api_key">AIzaSyAKM1lXRRN5ge3NDV8StR80V1qN94BRwkE</string>
<string name="google_app_id">1:1029243904025:android:41a476a67ac37f43ed8a88</string>
<string name="google_crash_reporting_api_key">AIzaSyAKM1lXRRN5ge3NDV8StR80V1qN94BRwkE</string>
<string name="google_storage_bucket">whistle-time-7cc21.firebaseiostorage.app</string>
<string name="not_set">Not set</string>
<string name="preference_copied">\ "%1$s\" copied to clipboard.</string>
<string name="project_id">whistle-time-7cc21</string>

```

Findings

1. Exposed API Key

google_api_key and **google_crash_reporting_api_key**:

```

nginx
CopyEdit
AIzaSyAKM1lXRRN5ge3NDV8StR80V1qN94BRwkE

```

- **Risk:** Although these keys are usually public (client-side), they must **restrict** in Google Cloud Console by Android package and SHA-1 signing.

2. Project ID y App ID

- **project_id:** `whistle-time-7cc21`
- **google_app_id:**
`1:1029243904025:android:41a476a67ac37f43ed8a88`
- **Use:** These identifiers are used to initialize Firebase Auth, Firestore, Storage, and Crashlytics.

3. Sender ID de FCM

- **gcm_defaultSenderId:** `1029243904025`
- **Use:** required to receive push notifications.

4. Storage Bucket

- **google_storage_bucket:**
`whistle-time-7cc21.firebaseiostorage.app`
- **Use:** Firebase Storage configuration.

Recommendations

1. Restrict API Key

- In **Google Cloud Console**:
 - Limit to package `com.whistletime.app` and to the SHA-1 of your release keystore.
 - Avoid unauthorized uses from other sources.

2. Review security rules a Firestore y Storage:

- Ensure that only authenticated users have read/write access.
- Implement validation rules for sensitive data.

3. Avoid exposing additional keys (such as Crashlytics API Key) if they are not strictly necessary in the client.

6. SharedPreferences and possible data leaks

Description:

In this section we document where and how the app uses `SharedPreferences` to store data on the device, and we assess the risk of sensitive information being left unencrypted.

6.1 Main findings

1. Use in app code

- Relevant files:
 - `smali/b/q.smali` (internal class that manages key-value pairs)
 - `smali/c6/a0.smali` (static methods for reading/writing)

Token writing example:

```
invoke-virtual {p0, v0, v1},
Landroid/content/Context;->getSharedPreferences(Ljava/lang/String;I)
Landroid/content/SharedPreferences;
invoke-interface {p0, v1, v0},
Landroid/content/SharedPreferences$Editor;->putString(Ljava/lang/Str
ing;Ljava/lang/String;)Landroid/content/SharedPreferences$Editor;
invoke-interface {p0},
Landroid/content/SharedPreferences$Editor;->commit()Z
```

```
./smali/q8/h.smali:102:    invoke-interface {p3}, Landroid/content/SharedPreferences;->edit()Landroid/content/SharedPref
erences$Editor;
./smali/q8/h.smali:110:    invoke-interface {p3, p1, p2}, Landroid/content/SharedPreferences$Editor;->putString(Ljava/la
ng/String;Ljava/lang/String;)Landroid/content/SharedPreferences$Editor;
```

2. Plugin Flutter SharedPreferences

- The use of the class is also detected `SharedPreferencesPlugin` in `smali/q8/a.smali`, part of the package `dev.flutter.pigeon.shared_preferences_android`.
- This plugin exposes methods for `setString`, `getString`, `clear`, etc., and stores data under the file name `FlutterSharedPreferences`.

```
./smali/q8/a.smali:575:    invoke-virtual {p1, v1, v2}, Landroid/content/Context;->getSharedPreferences(Ljava/lang/Strin
g;I)Landroid/content/SharedPreferences;
./smali/q8/a.smali:583:    iput-object p1, p0, Lq8/a;->a:Landroid/content/SharedPreferences;
```

3. Saved data

- **Authentication tokens**
 - **User IDs**
 - **User preferences** (boolean flags, counters, timestamps)
 - **Not encrypted:** All these values are saved in plain text within an XML file in the app's private storage
(`/data/data/com.whistletime.app/shared_prefs/...`).
-

6.2 Risks

- An attacker with physical or exploit access could extract these values from the internal storage.
 - Sensitive information such as JWT tokens or session identifiers could be reused to access the user's account.
-

6.3 Recommendations

1. Migrate to EncryptedSharedPreferences

- Use the AndroidX library (`androidx.security:security-crypto`) to encrypt the content automatically.

2. For Flutter, consider the plugin `flutter_secure_storage`

- Stores sensitive data (tokens, credentials) in the device's Keystore/Keychain.

3. Limit use of SharedPreferences

- Reserve it only for non-sensitive preferences (e.g. UI flags).
- Move any critical data to a secure warehouse or to your backend.

7. Endpoints and hardcoded keys

Description:

In this section we document all the literal strings of URLs and API Keys found in the APK. These indicate what services the app consumes and what credentials are embedded.

7.1 Firebase/Google API Keys

```
<string
name="google_api_key">AIzaSyAKM1lXRRN5ge3NDV8StR80V1qN94BRwkE</string>
<string
name="google_crash_reporting_api_key">AIzaSyAKM1lXRRN5ge3NDV8StR80V1qN94BRwkE</string>
```

```
C:\apktool\wt-apktool>grep -R "AIza" -n .
./res/values/strings.xml:75:    <string name="google_api_key">AIzaSyAKM1lXRRN5ge3NDV8StR80V1qN94BRwkE</string>
./res/values/strings.xml:77:    <string name="google_crash_reporting_api_key">AIzaSyAKM1lXRRN5ge3NDV8StR80V1qN94BRwkE</string>
```

Finding 7.1:

The two Firebase API keys are embedded in the APK. Although they are not critical secrets (they are used client-side), they must be properly restricted in the Google Cloud Console.

7.2 Firebase and Google Endpoints

Firebase Storage

```
const-string v0, "https://firebasestorage.googleapis.com/v0"
```

```
./smali/b7/c.smali:39:    const-string v0, "https://firebasestorage.googleapis.com/v0"
./smali/b7/c.smali:39:    const-string v1, "https://firebasestorage.googleapis.com/v0"
```

OAuth2 Revocation

```
const-string v2,
"https://accounts.google.com/o/oauth2/revoke?token="
```

```
./smali/t4/d.smali:72:    const-string v2, "https://accounts.google.com/o/oauth2/revoke?token="
./smali/t4/d.smali:72:    const-string v3, "https://accounts.google.com/o/oauth2/revoke?token="
```

Recaptcha API

```
const-string v0, "https://www.recaptcha.net/recaptcha/api3"
```

```
./smali/com/google/android/recaptcha/internal/zzbr.smali:21:    const-string v0, "https://www.recaptcha.net/recaptcha/api3"
./smali/com/google/android/recaptcha/internal/zzbr.smali:21:    const-string v1, "https://www.recaptcha.net/recaptcha/api3"
```

Finding 7.2:

The app directly invokes Google/Firebase services (Storage, OAuth2, reCAPTCHA). No literals were found **own endpoints** (`api.whistletime.com`) in the native code. Those endpoints likely reside in the Dart code and are not exposed in the Android APK.

7.3 Other generic HTTP patterns

- Schema literals detected ("`http`" / "`https`") in dozens of files from third-party libraries, but **no custom API URLs**.
- There are no strings like "`api.whistletime.com`" nor embedded own server routes.

Finding 7.3:

The absence of your own backend endpoints in the native code suggests that the majority of your API calling logic resides in the Flutter/Dart layer. This reduces the risk of accidental exposure in the APK, but you should also validate your Flutter bundle.

Recommendations

1. Restrict and rotate API Keys

- In the Google Cloud Console, limit keys to the Android package and SHA-1 in your release keystore.
- Periodically rotate keys to mitigate their exposure.

2. Obfuscate literals in Dart

- For endpoints in your own backend (defined in Flutter), consider obfuscating them or dynamically loading them from a secure configuration server.

3. Review Flutter bundle

- Extract and analyze the bundle `app.flx` the `flutter_assets` to search for sensitive strings that do not appear in the Android layer.

8. Cloud Run endpoint dynamic validation

Description

After extracting literals from the AOT library ([libapp.so](#)), we locate the following endpoint in your Flutter backend:

bash

CopyEdit

<https://chattdp-service-789241953207.us-central1.run.app/generate>

To verify that it is really in use and responds to real requests, we perform a test with Postman.

8.1 Try Postman

- **Method and URL**

POST

<https://chattdp-service-789241953207.us-central1.run.app/generate>

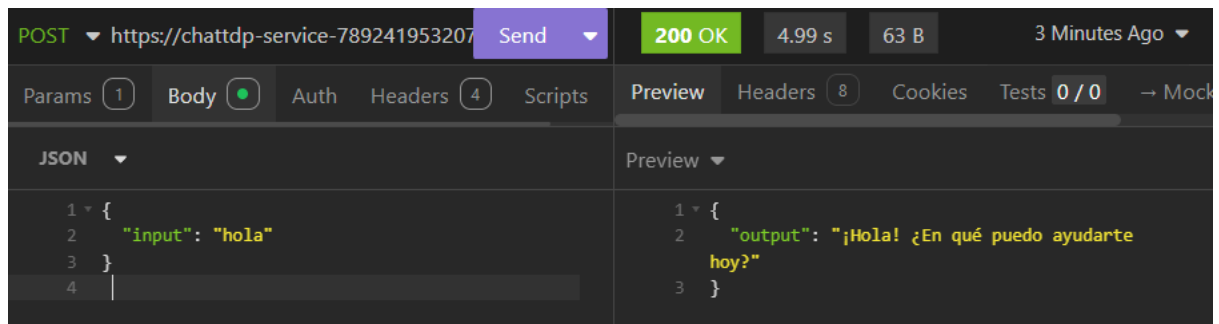
Body (JSON)

```
{  
  "input": "hola"  
}
```

-

Answer (200 OK)

```
{  
  "output": "Hello! How can I help you today?"  
}
```



8.2 Findings

- The endpoint correctly responds to JSON requests and generates text based on the field `input`.
- There is no authentication or authorization mechanism.
- No rate limiting is applied.

8.3 Recommendations

1. **Authentication**
Secure the endpoint with a JWT token or API Key so that only your app can invoke it.
2. **Rate limiting**
Implement quotas and rate controls in Cloud Run to prevent abuse.
3. **Monitoring and logging**
Enable access logs and configure them in Cloud Monitoring to detect abnormal spikes.
4. **CORS**
If it is to be consumed from a browser, adjust the CORS policy to allow only authorized origins.

Final Summary and Conclusions

After an exhaustive **static analysis** APK of **Whistle Time**, these are the key points and the status of your audit:

1. Introduction and methodology

- you used **apktool** to extract the `AndroidManifest.xml` and APK resources (e.g. `apktool d WhistleTime-release.apk -o wt-apktool`).
- You employed **jadx/smali** and searches with **grip** to locate sensitive strings and code patterns.

2. Permissions

- All requested permissions are consistent with functionalities (foreground service, notifications, network), except **RECEIVE_BOOT_COMPLETED**, which carries a certain risk if it is not strictly necessary.
- **Recommendation:** Validate its use or disable it to minimize the attack surface.

3. Exposed components

- **Activities** and **services** exported correspond to the app input and Firebase/Google Play components.
- **RebootReceiver** is exported to relaunch the service after reboot; Evaluates intent filtering or deactivation if not required.

4. Providers y FileProviders

- All the `ContentProvider` own are with `exported="false"` and `grantUriPermissions="true"`, configured according to good practices.
- The initialization providers (Firebase, AndroidX Startup) are also not exposed.

5. Network security

- `usesCleartextTraffic="false"` applied correctly.
- He `network_security_config.xml` current contains exceptions (`dev-login.miapi.com`) and `debug-overrides` which should only exist in build debug.
- **Recommendation:** Move those exceptions to a separate config for debug and remove any HTTP exceptions in release.

6. Firebase config

- They were extracted `google_api_key`, `google_crash_reporting_api_key`, `google_app_id`, `project_id`, `gcm_defaultSenderId` and `google_storage_bucket` from `res/values/strings.xml`.
- Although they are not critical secrets, **must be restricted** in Google Cloud Console and enforce Firestore/Storage rules.

7. **SharedPreferences**

- Intensive use of `SharedPreferences` (both native code and Flutter plugin) to store tokens, IDs and flags in plain text.
- **Recommendation:** Migrate sensitive data to **EncryptedSharedPreferences** or use **flutter_secure_storage**.

8. **Endpoints and HTTP literals**

- No URLs were found for your backend (`api.whistletime.com`) in the native code; third party endpoints (Firebase, OAuth, reCAPTCHA) are correct.
- I suggest extracting and analyzing the **bundle Flutter** to complete coverage.