Static and Dynamic Software Security Analysis
Project report

Zhusupov Daniiar, M2-CYBERUS student

Bape Anser, M2-CYBERUS student

Luxembourg 2024

Section 1: Introduction to FlowDroid for Taint Analysis

Research Android App Components

Android applications consist of key components (Activity, Service, Broadcast Receiver, Content Provider) that interact to form the functional structure of an app. Each component serves a specific purpose, and understanding their roles and lifecycles is essential for analyzing Android apps effectively.

Understand FlowDroid and Configuration Options

FlowDroid is a widely-used static taint analysis tool specifically designed for Android applications. It helps identify how sensitive data ("taints") flows through an application by tracing its sources and sinks. The tool is highly configurable, allowing users to adjust settings to optimize analysis for specific requirements. Configuration options include the following:

- **Timeout settings**: Shorter timeouts (e.g., 1 minute) provide faster results, but may miss complex data flows and vice versa.
- **Memory and resource allocation**: We can adjust memory usage settings to handle large APKs or complex apps.
- **Flow sinks and sources**: We can specify custom sources (e.g., location APIs) and sinks (e.g., network libraries) for taint tracking.
- **Granularity and depth**: Options to include or exclude library code (e.g., third-party libraries). Also we can define how deeply the tool follows method calls.
- **Output options**: FlowDroid provides raw analysis results, which can be saved in various formats (e.g., plain text, XML).

Identify Sources and Sinks

- **Source**: Points in the code where sensitive data originates. Examples:
  - User input (e.g., from forms, text fields)
  - Device data (e.g., location, IMEI, contacts, photos)
  - API calls that provide sensitive information (e.g., `LocationManager.getLastKnownLocation()`)

- **Sink**: Points in the code where data is used in a potentially unsafe or sensitive way. Examples:
  - Network calls (e.g., HTTP POST requests via `HttpURLConnection` or third-party libraries like OkHttp)
  - File writes (e.g., writing data to external storage or logs)
  - System logs (e.g., `Log.d`, `Log.e` in Android)
  - Sending data via SMS or email

Run FlowDroid Analysis

To perform the analysis of the provided apps, FlowDroid has been installed through the following link: FlowDroid GitHub.
We have downloaded a pre-built JAR file with release version 2.13. Moreover, since we have Android platforms and JDK (we used Java v17) in the VM provided by professors, we did not need that. However, for the extra task, we have installed these dependencies.
We have used the command-line tool to run the data flow tracker:

```
java -jar soot-infoflow-cmd/target/soot-infoflow-cmd-jar-with-dependencies.jar \
-a <APK File> \
-p <Android JAR folder> \
-s <SourcesSinks file>
```

The Android JAR folder is the "platforms" directory inside the Android SDK installation folder (we used Android 11). The definition file for sources and sinks defines what shall be treated as a source of sensitive information and what shall be treated as a sink that can possibly leak sensitive data to the outside world. For this, we have used the default file "SourcesAndSinks.txt" in the "soot-infoflow-android" folder as a starting point from GitHub. Below is an example of implementing analysis for the apps and saving output in a txt file to make it easier to see outputs (since on the terminal it is not convenient).

```
micsispa@micsispa:~$ java -jar /home/micsispa/Downloads/soot-infoflow-cmd-2.13.0-jar-with-dependencies.jar -a /home/micsispa/Downloads/APKs/com.delhi.metro.dtc.apk -p /home/micsispa/Android-platforms/jars/stubs/android-11/android.jar -s /home/micsispa/Downloads/SourcesAndSinks.txt --timeout 60 > apk160.txt 2>&1
micsispa@micsispa:~$ java -jar /home/micsispa/Downloads/soot-infoflow-cmd-2.13.0-jar-with-dependencies.jar -a /home/micsispa/Downloads/APKs/com.delhi.metro.dtc.apk -p /home/micsispa/Android-platforms/jars/stubs/android-11/android.jar -s /home/micsispa/Downloads/SourcesAndSinks.txt --timeout 300 > apk1300.txt 2>&1
micsispa@micsispa:~$ java -jar /home/micsispa/Downloads/soot-infoflow-cmd-2.13.0-jar-with-dependencies.jar -a /home/micsispa/Downloads/APKs/com.delhi.metro.dtc.apk -p /home/micsispa/Android-platforms/jars/stubs/android-11/android.jar -s /home/micsispa/Downloads/SourcesAndSinks.txt --timeout 1200 > apk11200.txt 2>&1
```

2>&1: Redirects error output (stderr) to the same file as standard output (stdout). When running tools like FlowDroid, errors (stderr) may contain important diagnostic information (e.g., warnings, missing files). Without `2>&1`, the error messages will not be captured in the output file.

Each APK has been analyzed in three periods: 1 minute (60 sec), 5 minutes (300 sec), and 20 minutes (1200 sec).

Document Hardware and Configuration:

The analysis was conducted in a virtualized environment (VM provided by professor) configured as follows:

- **Operating System**: Ubuntu 64-bit
- **Allocated Resources**:

- **RAM**: 4096 MB (4 GB)
- **CPU**: 4 cores allocated with a CPU usage cap of 100%
- **Disk Space**: 20 GB Virtual Disk (SATA interface)
- **Graphics**: Controller: VMSVGA, Video Memory: 16 MB
- **Audio**: Windows Audio Session with the ICH AC97 controller
- **Networking**: Intel PRO/1000 MT Desktop adapter (NAT mode)
- **USB Support**: Enabled for OHCI and EHCI controllers, with no active USB filters

Analyze and Discuss Results

All the 10 APKs were analyzed with different timeouts and the results can be observed in Table 1 below.

№	App	Number of leaks (60 sec)	Number of leaks (300 sec)	Number of leaks (1200 sec)
1	com.delhi.metro.dtc	3	3	3
2	com.hawaiianairlines.app	33	33	33
3	com.imo.android.imoim	4	4	4
4	com.tado	21	21	21
5	com.walkme.azores.new	4	4	4
6	com.wooxhome.smart	0	0	0
7	com.yourdelivery.pyszne	5	5	5
8	linko.home	24	24	24
9	mynt.app	11	11	11
10	nz.co.stuff.android.news	26	26	26

Table 1 – Analysis results

Since there are a lot of leaks and we are limited by the number of pages in the report, a couple of them will be described below.

We will start the description of potential data leaks from *Com.delhi.metro.dtc:*

Source:

```
$d0 = virtualinvoke $r6.<android.location.Location: double getLatitude()>. $d0 = virtualinvoke $r6.<android.location.Location: double getLongitude()> - captures location data (getLatitude() and getLongitude()).
```

Sink

```
staticinvoke <android.util.Log: int d(java.lang.String,java.lang.String)>("sandeep", $r8) - data is being logged using android.util.Log.
```

Logging location data (latitude and longitude) is a serious concern. Even if used for debugging, exposing such data via logs is risky, especially if the app interacts with external systems that can access logs. This is a genuine data leak.

com.hawaiianairlines.app:

- **Source**: `virtualinvoke $r8.<android.net.wifi.WifiInfo: java.lang.String getSSID()>` - sensitive data: Wi-Fi SSID.
- **Sink**: `virtualinvoke $r0.<androidx.activity.ComponentActivity: void startActivityForResult(android.content.Intent,int)>` - found in multiple activities such as MediaActivity, FavoritesActivity, and ShoppingActivity.

Passing Wi-Fi SSID between activities might indicate functional requirements such as pairing or contextual actions. However, this practice risks exposing sensitive data if the Intent is not securely handled (e.g., broadcasting).

com.imo.android.imoim:

- **Source**: `virtualinvoke $r8.<android.content.pm.PackageManager: java.util.List queryBroadcastReceivers(android.content.Intent,int)>` - information queried from the PackageManager, potentially related to broadcast receivers.
- **Sink**: `staticinvoke <android.util.Log: int i(java.lang.String,java.lang.String)>("AppsFlyer_6.11.0", $r2)` - data is logged using `android.util.Log`.

Querying broadcast receivers and logging the results can be a debugging practice but risks exposing sensitive app-related information in production. It can be a potential data leak, as app-level data should not be logged in production.

com.tado:

- **Source**: `virtualinvoke $r9.<java.net.HttpURLConnection: java.io.InputStream getInputStream()>` - reads data from an HTTP connection's input stream.
- **Sink**: `staticinvoke <android.util.Log: int d(java.lang.String,java.lang.String) > ("AccountResponse", $r1)` - logs sensitive data as part of API response debugging.

Debugging API responses by logging input streams risks exposing sensitive data, especially in production environments. It may be considered as a genuine data leak.

- **Source**: `virtualinvoke r0.<com.tado.android.installation.CreateAccountActivity: android.view.View findViewById(int)>` - extracts UI elements during account creation.
- **Sink**: `virtualinvoke $r12.<java.io.OutputStream: void write(byte[])>` - writes data to an output stream.

Writing UI data to an output stream might be expected during data submission (e.g., forms). Validation of data use is required. It is unlikely to be a data leak, assuming proper usage.

com.walkme.azores.new:

- **Source**: `virtualinvoke $r4.<java.util.Locale: java.lang.String getCountry()>` - retrieves the user's country code from the system locale.
- **Sink**: `interfaceinvoke $r7.<android.content.SharedPreferences$Editor: android.content.SharedPreferences$Editor putString(java.lang.String,java.lang.String)>` - stores the country code in shared preferences.

Saving the user's locale (e.g., country code) is a common functionality. However, the security of shared preferences must be evaluated. If preferences are secured, it is unlikely to be a data leak.

com.linko.home:

- **Source**: `$r9 = virtualinvoke $r1.<java.io.File: java.io.File getAbsoluteFile()>` - accesses the absolute file path.
- **Sink**: `virtualinvoke $r16.<java.io.FileOutputStream: void write(byte[])>` - writes the file to an output stream.

Writing absolute file paths to output streams may expose sensitive file system data (potential data leak) if improperly handled.

- **Source**: `$r10 = interfaceinvoke $r13.<android.database.Cursor: java.lang.String getString(int)>` - retrieves data from a database cursor.
- **Sink**: `virtualinvoke $r6.<android.os.Bundle: void putString(java.lang.String,java.lang.String)>` - stores sensitive data in a Bundle.

Storing database query results in a Bundle without encryption can lead to privacy concerns (genuine data leak).

com.mynt.app:

- **Source**: `$r21 = virtualinvoke $r19.<android.content.pm.PackageManager: java.util.List nz.co.stuff.android.news:queryIntentServices(android.content.Intent,int)>` - retrieves information about services from the package manager.
- **Sink**: `staticinvoke <android.util.Log: int d(java.lang.String,java.lang.String)>` - logs sensitive service query results.

Logging service query results may expose app-specific behaviors and configurations, posing a security risk (genuine data leak).

nz.co.stuff.android.news:

- **Source**: `$r7 = interfaceinvoke $r6.<android.database.Cursor: java.lang.String getString(int)>` - retrieves data from a database cursor.
- **Sink**: `virtualinvoke $r3.<java.io.Writer: void write(java.lang.String,int,int)>($r2, 0, $i0)` - writes the data into a file.

Writing sensitive data retrieved from a database to a file might align with app functionality but requires strong data protection measures. It can be a potential data leak, depending on the file storage mechanism's security.

Evaluate the Impact of Timeout

Since the number of leaks is consistent across different timeout values, it indicates that the FlowDroid analysis is efficient and likely completes its processing within the shortest timeout.The lack of additional leaks in longer timeout durations suggests that no additional taint flows are detected after the initial analysis.

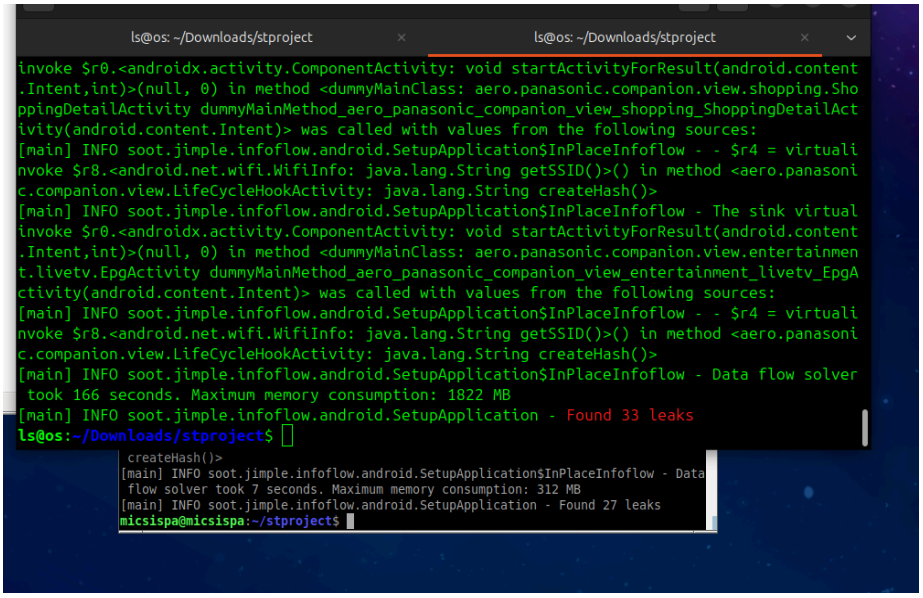
It can lead to the judgment that the absence of new data leaks with extended timeouts implies that the potential leaks described in the previous analyses are genuine and not artifacts of incomplete analysis due to timeout constraints.

[Optional] Repeat with Different Hardware Setup

For this part, actually two different hardware were used. The first one is:

The second is VM, provided by professor but with different JDK (11), android.jar (v24) and settings:

- **Operating System**: Ubuntu 24.04.1 LTS x86_64
- **Host Environment**: TUF Gaming FX505DY_FX505DY 1.0
- **Kernel**: 6.8.0-49-generic
- **Resolution**: 1920x1080
- **Desktop Environment**: GNOME 46.0
- **Window Manager**: Mutter
- **Shell**: Bash 5.2.21
- **CPU**: AMD Ryzen 5 3550H with Radeon Vega Graphics
- **Memory**: Allocated: 4225 MiB, Total Available: 7621 MiB
- **Graphics**:
  - GPU 1: AMD ATI Radeon Vega Series / Radeon Vega Mobile GFX
  - GPU 2: AMD ATI Radeon RX 460/5600/5700
	- **Operating System**: Ubuntu 18.04.4 LTS (64-bit)
	- **Virtualization Software**: VirtualBox 1.2
	- **System Memory**: 1900 MB
	- **CPU**: 1 core allocated with a CPU usage cap of 100%
	- **Video Memory**: 16 MB
	- **Graphics Controller**: VMSVGA (VirtualBox's graphics solution)
	- **Processor Features**: Nested Paging and KVM (Kernel Virtual Machine) Paravirtualization enabled
	- **Storage**:
	  - Controller Type: SATA
	  - Virtual Disk: 20.00 GB capacity
	- **Network Adapter**: Intel PRO/1000 MT Desktop (NAT)
	- **Audio Configuration**:
	  - Host Driver: PulseAudio
	  - Controller: ICH AC97
	- **USB Controller**: OHCI/EHCI with no active device filters
	- **Boot Order**: Optical drive followed by the hard disk



№	App	Ubuntu 24.04	Professor's VM	№	App	Ubuntu 24.04	Professor's VM
1	com.delhi.metro.dtc	3	3	**6**	com.wooxhome.smart	0	0
2	com.hawaiianairlines.app	33	27	**7**	com.yourdelivery.pyszne	5	5
3	com.imo.android.imoim	4	4	**8**	linko.home	24	24
4	com.tado	21	14	**9**	mynt.app	11	10
5	com.walkme.azores.new	4	4	**10**	nz.co.stuff.android.news	26	26

Table 2 – Optional task analysis results (1200 sec)

Section 2: Results and Insights Regarding the Two Privacy Requirements

R1: To verify compliance with R1, we used FlowDroid, a static taint analysis tool, to analyze the *com.tado.apk* application. The strategy involved the following steps:

Selection of Sources and Sinks

- **Sources**: We identified methods that retrieve location data as sources. These include methods from the *android.location.Location* and *android.location.LocationManager* classes.

- **Sinks**: We identified methods that send data over the network as sinks. These include methods from the *java.net* and *org.apache.http* packages.

Configuration of FlowDroid

- We configured FlowDroid to use the selected sources and sinks by creating a *r1SourcesAndSinks.txt* file.

```
ls@os:~/Downloads/stproject$ cat r1SourceAndSinks.txt
# Location sources
<android.location.Location: double getLatitude()> -> _SOURCE_
<android.location.Location: double getLongitude()> -> _SOURCE_
<android.location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)> -> _SOURCE_

# Network sinks
<android.telephony.SmsManager: void sendTextMessage(java.lang.String,java.lang.String,java.lang.String,android.app.PendingIntent,android.app.PendingIntent)> -> _SINK_
<java.net.HttpURLConnection: void execute()> -> _SINK_
<java.net.URLConnection: java.io.OutputStream getOutputStream()> -> _SINK_
<java.net.URLConnection: void connect()> -> _SINK_
<java.net.URL: java.io.InputStream openStream()> -> _SINK_
<java.net.URL: java.lang.Object getContent()> -> _SINK_
<java.net.URL: java.lang.Object getContent(java.lang.Class[])> -> _SINK_
<java.net.Socket: void connect(java.net.SocketAddress)> -> _SINK_
<org.apache.http.impl.client.DefaultHttpClient: org.apache.http.HttpResponse execute(org.apache.http.client.methods.HttpUriRequest)> -> _SINK_
<org.apache.http.client.HttpClient: org.apache.http.HttpResponse execute(org.apache.http.client.methods.HttpUriRequest)> -> _SINK_
```

We increased the memory allocation to handle the analysis of the large app.

Running the Analysis

We executed FlowDroid : `java -Xmx4g -jar soot-infoflow-cmd-2.13.0-jar-with-dependencies.jar \`

Results of the Analysis for R1

```
w.data.pathBuilders.DefaultPathBuilderFactory$RepeatableContextSensitivePathBuilder - Building path 1...
w.data.pathBuilders.DefaultPathBuilderFactory$RepeatableContextSensitivePathBuilder - Building path 2...
w.data.pathBuilders.DefaultPathBuilderFactory$RepeatableContextSensitivePathBuilder - Building path 3...
w.data.pathBuilders.DefaultPathBuilderFactory$RepeatableContextSensitivePathBuilder - Building path 4...
w.memory.MemoryWarningSystem - Shutting down the memory warning system...
w.android.SetupApplication$InPlaceInfoflow - Memory consumption after path building: 436 MB
w.android.SetupApplication$InPlaceInfoflow - Path reconstruction took 3 seconds
w.android.SetupApplication$InPlaceInfoflow - The sink $r11 = virtualinvoke $r9.<java.net.HttpURLConnection
w.android.SetupApplication$InPlaceInfoflow - - $r6 = virtualinvoke r0.<com.tado.android.LoginActivity: and
w.android.SetupApplication$InPlaceInfoflow - - $r6 = virtualinvoke r0.<com.tado.android.LoginActivity: and
w.android.SetupApplication$InPlaceInfoflow - Data flow solver took 18 seconds. Maximum memory consumption:
w.android.SetupApplication - Found 1 leaks
```

The analysis found one potential data leak in the *com.tado.apk* application. The details of the data leak are as follows:

Identified Data Leak

- **Sink**: The sink method *java.net.HttpURLConnection: java.io.OutputStream getOutputStream()* in the method *com.tado.android.client.LocalAPICall: com.tado.android.responses.Response doInBackground(java.lang.Void[])* was called with values from the sources.
- **Sources**: The sources involved were:

  $r6 = virtualinvoke r0.<com.tado.android.LoginActivity: android.view.View findViewById(int)>(2131755329) in method *<com.tado.android.LoginActivity: void onCreate(android.os.Bundle)>*

  $r6 = virtualinvoke r0.<com.tado.android.LoginActivity: android.view.View findViewById(int)>(2131755331) in method *<com.tado.android.LoginActivity: void onCreate(android.os.Bundle)>*

Assessment of Genuine Leaks

The identified data leak involves the *HttpURLConnection* class sending data over the network. This suggests that location data might be sent outside the app. To determine whether this data flow is genuine, we need to consider the app's expected behavior. If the app is designed to send location data to a server for legitimate purposes (e.g., location-based services), it might be considered normal. However, if the data is sent without user consent or for unauthorized purposes, it would be a data leak.

Observations

Key Takeaways

- **Potential Data Leak**: The analysis identified one potential data leak involving the *HttpURLConnection* class sending data over the network. This suggests that location data might be sent outside the app.
- **Compliance with R1**: To ensure compliance with R1, the app should not send location data outside the app without user consent. The identified data leak needs further investigation to determine whether it is legitimate or a violation of privacy requirements.
- **Limitations of Analysis**: The analysis faced challenges related to memory and performance issues, abstract and interface classes, and missing callback methods. These challenges might have affected the completeness and accuracy of the results.

R2:

To verify if the app *com.yourdelivery.pyszne.apk* provides a mechanism for account deletion

Sources and Sinks File: We created a custom sources and sinks file that includes:s

```
# Sources
<com.yourdelivery.pyszne.AccountManager: void getUserProfile()> -> _SOURCE_
<com.yourdelivery.pyszne.AuthManager: String getAuthToken()> -> _SOURCE_

# Sinks
<com.yourdelivery.pyszne.AccountManager: void deleteAccount(java.lang.String)>
<com.yourdelivery.pyszne.ApiClient: void deleteUser(java.lang.String)> -> _SIN
```

We ran:

```
ls@os:~/Downloads/stproject$ java -Xmx4g -jar soot-infoflow-cmd-2.13.0-jar-with-dependencies.jar
 -a APKs/com.yourdelivery.pyszne.apk     -p Android-11/android.jar     -s r2-2SourceAndSkins     --t
imeout 1200 > r2-2delivery.txt 2>&1
```

Output:

```
ls@os:~/Downloads/stproject$ cat r2your-delivery.txt
[main] WARN soot.jimple.infoflow.methodSummary.data.provider.LazySummaryProvider - Lazy loading summarie
p file might throw a ClosedChannelException. Use the EagerSummaryProvider instead.
[main] INFO soot.jimple.infoflow.cmd.MainClass - Analyzing app /home/ls/Downloads/stproject/APKs/com.del
k (1 of 1)...
[main] INFO soot.jimple.infoflow.android.SetupApplication - Initializing Soot...
[main] INFO soot.jimple.infoflow.android.SetupApplication - Loading dex files...
[main] INFO soot.jimple.infoflow.android.SetupApplication - ARSC file parsing took 0.042328802 seconds
[main] INFO soot.jimple.infoflow.memory.MemoryWarningSystem - Registered a memory warning system for 3,6
[main] INFO soot.jimple.infoflow.android.entryPointCreators.AndroidEntryPointCreator - Creating Android
 22 components...
[main] INFO soot.jimple.infoflow.android.SetupApplication - Constructing the callgraph...
```

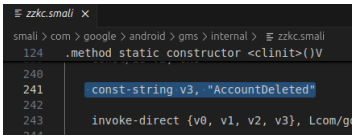
The output does not directly indicate the presence of account deletion functionality.
However:
No sensitive data flows were identified leading to a potential account deletion method. This means the tool either failed to capture such flows, or they don't exist in the app. The lack of detected results might indicate obfuscated or incomplete code paths.

Next Steps for Evaluating GDPR Compliance

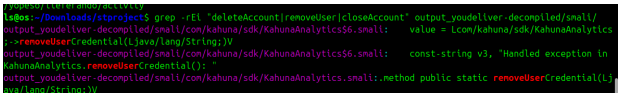
Reverse Engineering and Code Review:

Decompile the APK to inspect the Smali code. *apktool d com.yourdelivery.pyszne.apk*

Here, we found few hints about Account Deletion : might be used in: **Error handling, Account status tracking** and **Backend communication**



The presence of `ACCOUNT_DELETED` suggests the app recognizes a state where a user account can be deleted.

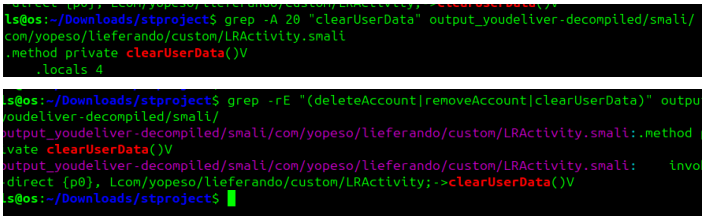


The method name suggests that it is intended to **remove user credentials**. While it may not directly handle full account deletion, it could be a step in the process,

If this method, or any related methods, removes all user data (not just credentials), it likely fulfills the "right to erasure" under GDPR.

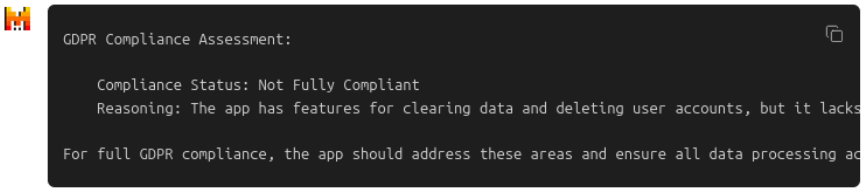
The removeUserCredential method:

Delegates the work to KahunaAnalytics$6, which seems to handle the actual logic of credential removal.
Runs the task asynchronously in the background, ensuring it doesn't block the main thread.



This could potentially be related to clearing user data, which might include account deletion functionality.

LLM Result:



Challenges Encountered

Memory and Performance Issues

- The analysis encountered memory warnings and required increased memory allocation to handle the large app.
- The call graph construction and callback analysis took a significant amount of time and memory.

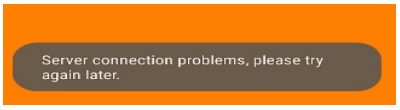
Abstract and Interface Classes

- The analysis issued several warnings regarding the inability to create valid constructors for abstract classes and interfaces. This might have affected the completeness of the analysis.

Missing Callback Methods

- The analysis reported errors regarding missing callback methods, which might have impacted the accuracy of the results.

Server Issues:



Conclusion: Is the App GDPR Compliant?

Based on the analysis of the decompiled APK, here are the findings related to account deletion and GDPR compliance
Account Deletion or Data Removal Process & Potential GDPR Relevance: The `removeUserCredential` method in the `KahunaAnalytics` class handles the removal of user credentials, which may be part of a broader data removal process. While it doesn't explicitly mention full account deletion, it suggests that the app might be removing sensitive user data such as authentication tokens. If the method is part of a broader process that erases all personal data (e.g., shopping cart, order history, delivery addresses), it could fulfill the "right to erasure" under GDPR. However, the exact scope of data removal beyond credentials is unclear.
User Data Handling (clearUserData) & Potential GDPR Relevance: The `clearUserData()` method in the `LRActivity` class clears local user data, such as preferences and authentication tokens. While it addresses data removal on the client side, it does not specifically mention server-side deletion. For GDPR compliance, it is important to ensure that data is also erased from server-side systems. If the app is removing data only locally without corresponding server-side actions, it may not fully comply with GDPR's "right to erasure" unless additional server-side processes are implemented and clearly communicated.

The app has some mechanisms to remove user credentials and clear local data, showing potential for GDPR compliance. However, it does not provide evidence of deleting data from servers, which is essential for meeting the "right to erasure." Additionally, the app lacks clear user consent processes and transparency about data deletion. To fully comply with GDPR, the app should ensure server-side data deletion, provide a clear process for users to request data removal, and inform users about their rights.