Security report on Mentorship System

by Bartek Pacia Google Code-in 2019

App security is one of such topics that can be ignored for a long time, but when problems related to it arise, effects are often devastating. Leaks of our users' data or tamperation of our API keys is not what we ever want to face. That's why we, as pragmatic developers, should deeply care about securing sensitive data.

In this document, I'll list best practices recommended by Official Android Documentation and verify whether Mentorship Android app follows them. If yes, I'll give code examples and if no, I'll suggest possible solutions.

I use following colors to denote status of usage of particular practices in Mentorship Android:

- Green We conform with the best practice
- Grey The best practice doesn't apply to Mentorship Android
- Red We don't conform with the best practice

Terms "the app", "our app", etc. refer to Systers' Mentorship Android application.

1. Enforce secure communication

1.1. Use implicit intents and non-exported content providers

Mentorship Android doesn't use Intents for anything but in-process communication, mainly launching new Activities. Therefore, this point doesn't apply currently (but it will when e.g profile picture taking feature is implemented).

1.2. Ask for credentials before showing sensitive information

Mentorship Android doesn't currently collect any kind of sensitive information. The only place where potentially dangerous action happens is Change Password functionality. In this case, an Alert Dialog is shown. It asks the user to enter the old password to confirm their identity. Passwords are *dotted* to protect them from peepers.

1.3. Apply network security measures

Our current backend hosted on AWS Elastic Beanstalk doesn't support

HTTPS, which is a serious security concern. In order to work with the unsecured backend, Mentorship Android has to use **android:usesCleartextTraffic="true"**. This basically means that data of out users, including passwords, travel back and forth unencrypted. It's very undesirable behavior even on development stage, because some users may use their real credentials to login to Mentorship System.

```
7       <application
8           android:name=".MentorshipApplication"
9           android:allowBackup="true"
10          android:icon="@mipmap/ic_launcher"
11          android:label="@string/app_name"
12          android:roundIcon="@mipmap/ic_launcher_round"
13          android:supportsRtl="true"
14          android:usesCleartextTraffic="${usesCleartextTraffic}"
```

AndroidManifest.xml. We see reference to usesCleartextTraffic value in build.gradle

```
20      buildTypes {
21          release {
22              minifyEnabled false
23              proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
24              manifestPlaceholders = [usesCleartextTraffic:"false"]
25          }
26          debug {
27              manifestPlaceholders = [usesCleartextTraffic:"true"]
28          }
29      }
```

app/build.gradle file. Cleartext traffic is enabled in debug mode and disabled in release. As of now, all users use debug mode.

Solution: Enable HTTPS support for backend. This would involve getting an SSL certificate, which about $50/year[1].

1.4. Use WebView objects carefully

Mentorship Android doesn't utilize WebViews.

2. Provide the right permissions

Mentorship Android doesn't make use of any permission except besides **android.permission.INTERNET**. Therefore, this section doesn't apply currently.

[1] Based on: https://www.ssl.com/certificates/basicssl/buy

3. Store data safely

3.1. Store private data within internal storage

The only information our app currently stores on-device is user's JWT token. PreferenceManager is responsible for taking care of managing it (i.e persisting it upon a successful login and deleting the token when the user logs out). However, after some research, it turns out that token is stored in plaintext using SharedPreferences. It's a serious security flaw and needs to be fixed.

```kotlin
class PreferenceManager {

    companion object {
        const val APPLICATION_PREFERENCE = "app-preferences"
        const val AUTH_TOKEN = "auth-token"
    }

    private val context: Context = MentorshipApplication.getContext()
    private val sharedPreferences: SharedPreferences = context.getSharedPreferences(
            APPLICATION_PREFERENCE, Context.MODE_PRIVATE)

    /**
     * Saves the authorization token to SharedPreferences file.
     * @param authToken String which is the authorization token
     */
    @SuppressLint("ApplySharedPref")
    //Cannot use .apply(), it will take time to save the token. We need token ASAP
    fun putAuthToken(authToken: String) {
        sharedPreferences.edit().putString(AUTH_TOKEN, "Bearer $authToken").commit()
    }

    val authToken: String
        get() = sharedPreferences.getString(AUTH_TOKEN, "")

    /**
     * Clears all the data that has been saved in the preferences file.
     */
    fun clear() {
        sharedPreferences.edit().clear().apply()
    }
}
```

app/src/main/java/org/systers/mentorship/utils/PreferenceManager.kt. Lack of encryption is easy to see.

Solution: I'd suggest encrypting the key using AES. Then encrypt AES secret key with RSA and store RSA key in Keystore.

I got this idea from Flutter Secure Storage plugin. This approach is very safe, it would practically make any attempts of token tamperation futile.

3.2. Use external storage cautiously

The app doesn't use external storage to store any kind of data.

3.3. Store only non-sensitive data in cache files

The app doesn't cache any data.

3.4. Use SharedPreferences in private mode

We comply with this rule (look at PreferenceManager.kt screenshot above).

4. Keep services and dependencies up-to-date

4.1. Check the Google Play services security provider

We don't use Google Play Services, so it's not a concern. However, when we decide to add Google Sign In (as stated in *Future Ideas* section), this topic will come back.

4.2. Update all app dependencies

Almost all our dependencies are outdated, most by more than 6 months. This doesn't create security issues, but makes it very difficult to follow latest, best practices from the world of Android development. For example, extension functions enabling better coroutine support in ViewModel (like viewModelScope()) aren't available.
This issue has been already raised in this PR.