COM 402 exercises 2025, session 9: Mobile Security

Exercise 9.1

You and an app developer agree that native libraries should be used with care in Android apps. As the evening continues, the developer tells you that a couple of months ago, she found a very similar app to one of her apps in the Google Play Store. In fact, the two apps look almost identical. Your blood is pumping faster. This might be a republishing scheme!

Explain to the developer what a republishing scheme is and why it is possible on Android.

Exercise 9.2

What are the risks of SSL pinning, and how do you mitigate them? You are given the following Kotlin snippet:

```
private fun createCertificatePinner(): CertificatePinner? {
    return if (BuildConfig.DEBUG) {
        // Allow proxy tools like Charles in debug
        null
    } else {
        CertificatePinner.Builder()
            .add("api.myapp.com", PRODUCTION_PIN)
            .build()
    }
}
```

Exercise 9.3

Your certificate is expiring tomorrow, but you have 100k users on an old app version. How do you handle certificate rotation?"

Exercise 9.4

You are building an Android app that integrates multiple third-party APIs.

Answer the following:

1. Identify at least three security weaknesses (key storage, reverse engineering, user data exposure).
2. Explain how attackers can extract these secrets using tools like `APKTool` or `JADX`.
3. Propose a secure redesign that removes hardcoded keys and encrypts local data.
4. Discuss why obfuscation is insufficient and when client-side secrets are acceptable.

The following code is currently used in production:

```
object ApiConfig {
    const val MAPS_API_KEY = "AIzaSyXXXXXX-Your-Real-Key"
    const val ANALYTICS_API_KEY = "UA-123456-7"
}

fun getUserData(context: Context): String {
```

```
    // Store user token in SharedPreferences
    val prefs = context.getSharedPreferences("user_prefs", Context.MODE_PRIVATE)
    return prefs.getString("user_token", "") ?: ""
}
```

Exercise 9.5

App repackaging is a major attack vector: attackers re-sign and redistribute a tampered APK.

1. Describe at least three technical mitigations against repackaging.
2. Consider the following snippet:

   Identify the weakness and propose an improved design.

```
public class MyApp extends Application {
    @Override
    public void onCreate() {
        super.onCreate();

        String expectedSignature =
            "4cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824";
        String currentSig = getMyAppSignature();
        if (!expectedSignature.equals(currentSig)) {
            System.exit(1);
        }
    }
}
```

```
public class MyApp extends Application {
    static {
        System.loadLibrary("appsec");
    }

    private native String getExpectedSignatureHash();

    @Override
    public void onCreate() {
        super.onCreate();

        String expected = getExpectedSignatureHash();
        String current = computeApkSignatureHash();

        if (current == null || !current.equals(expected)) {
            disableApp();
        }
    }
}
```