

Mobile Security

Week 4: Static tooling

Static Analysis

Static analysis

Static analysis (also known as static code analysis or source code analysis) employs tools to examine program code in search of application coding vulnerabilities, back doors, or other malicious code that might provide hackers access to sensitive corporate data or consumer data.

Static analysis

Search for interesting data (passwords, URLs, API endpoints, API keys, encryption, tokens, ...)

- Check if the application is in debug mode and try to "exploit" it
 - Can I start the postLogin screen without logging in?
- Is the application saving data locally or external?
 - What is actually stored when the data is locally?
 - Can you bypass the data checked online?

Static analysis

Static analysis evaluates an APK's internal structure—such as Java/Kotlin code, Smali code, bytecode, dependencies, and manifest files—using automated tools or manual techniques.

This method is crucial for spotting issues like insecure storage, hardcoded secrets, improper permissions, and vulnerable APIs before deployment

Static analysis

- Exported Activities
 - Activities is the UI in Android
- Content Providers
 - Content providers in Android are a key component that manage access to a central repository of data.
- Exposed services
 - Background services

Static analysis

- Broadcast Receivers
 - Broadcast Receivers in Android are components that allow applications to listen for and respond to broadcast messages from the Android system or other apps.
- URL Schemes
 - URL Schemes in Android are custom URI schemes that allow apps to be launched and interacted with through specially formatted URLs.

Logcat

Logcat

Logcat is a command-line tool that dumps a log of system messages, including stack traces when the device throws an error and messages that you have written from your app with the Log class.

`adb logcat` gives all the log information back.

The `--help` option gives all information about this command.

Logcat

- Find the PID

```
adb shell ps -ef | grep appname
```

- Grep on PID

```
adb logcat | grep 5021
```

- Find the specific log messages for this application

howest
hogeschool

APK

What is an APK?

An APK file is an archive that usually contains the following files and directories:

- META-INF directory
- RES directory
- lib folders
- AndroidManifest.XML
- .dex classes
- .properties files

META-INF Directory

The META-INF directory contains metadata about the APK, including cryptographic signatures and certificates used to verify the integrity and authenticity of the app during installation. This directory plays a key role in securing the APK and ensuring it has not been tampered with.

RES Directory

The res directory holds app resources in multiple subfolders, such as:

- anim/ & animator/: Animation XML & property files.
- color/: Color state XMLs.
- drawable/: Images and drawable resources (can be PNG, XML, etc.).
- layout/: XML files defining UI layouts.
- mipmap/: Launcher icons for various screen densities.
- values/: XML files (strings, styles, arrays).
- xml/: Miscellaneous XML configuration files.

lib Folders

The lib directory contains native shared libraries, typically organized by architecture (like armeabi-v7a, arm64-v8a, x86).

These libraries are usually dynamic/shared object files (.so files) often written in C or C++ to provide platform-specific functionality.

AndroidManifest.XML

AndroidManifest.xml is an essential file that declares app components (activities, services, receivers, content providers), required permissions, hardware/software features, app metadata, and entry points.

The manifest is necessary for Android OS and Google Play to properly install, run, and secure the app.

.dex classes

classes.dex files contain the compiled app code in Dalvik Executable format.

These files are what the Android Runtime (ART/Dalvik) loads and executes.

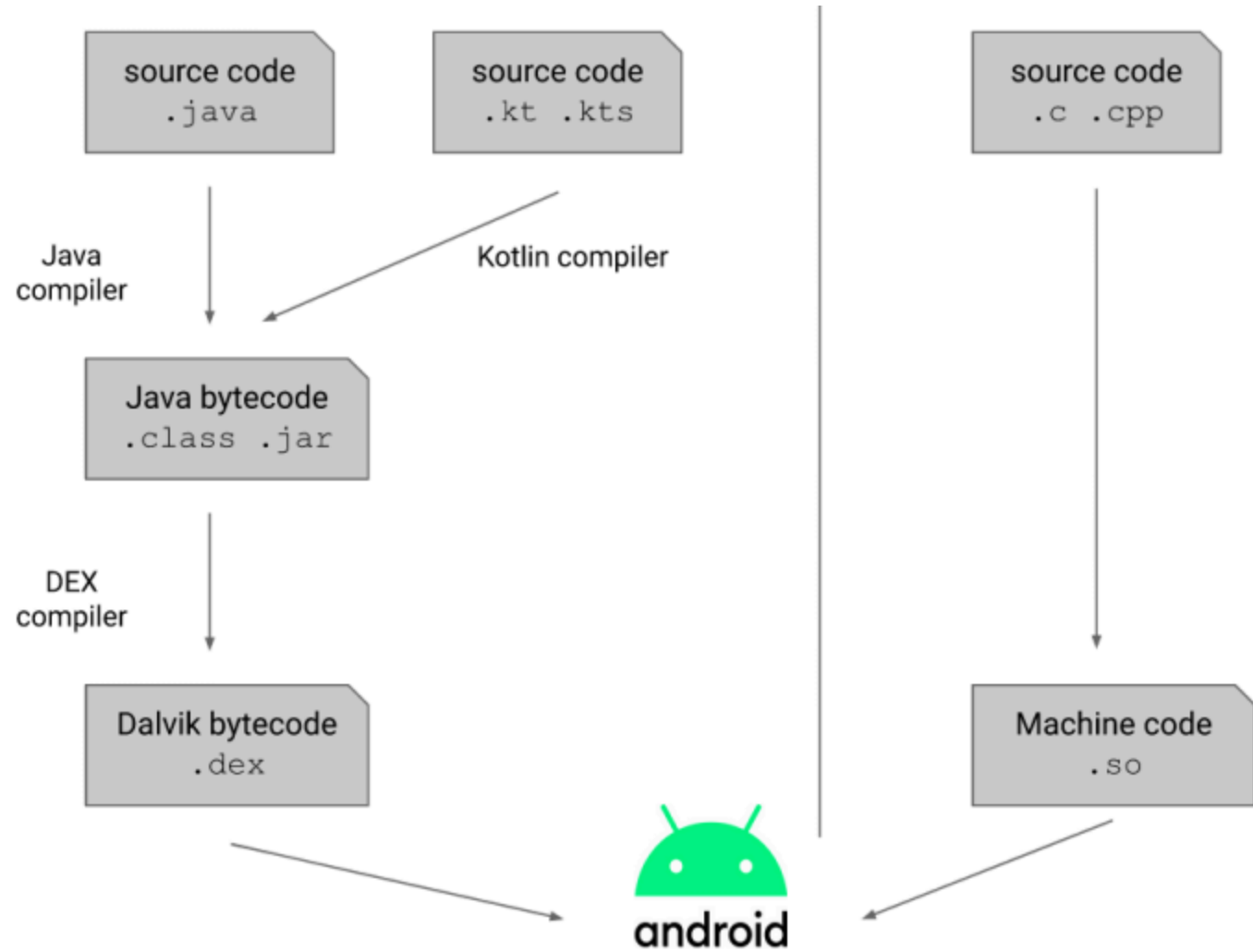
Multiple DEX files may exist in complex apps; they represent converted Java/Kotlin code that runs on Android devices.

.properties Files

.properties files are plain-text configuration files using key-value pairs, often used to store environment variables, settings, or app configuration parameters required at runtime or build time.

While not always present in every APK, when included, they permit externalizing configuration from code

From code to APK



Manifest

- Package name
 - Unique identifier ➡ two apps cannot have the same package name on one device
- Permissions
- App components
 - Activities
 - Services
 - Broadcast Receivers
 - Content Providers

Structure APK

After building an application an APK is generated (Android Application Package).



application.zip

Structure APK

```
$ unzip -d application application.apk
$ ls -l application
AndroidManifest.xml
META-INF/
classes.dex, classes2.dex, ...
lib/
assets/
```

- .dex ➡ Dalvik bytecode
- lib ➡ Native libraries
- assets ➡ Read-only data

Structure APK

Tools are needed to make the data readable (decode the data).

Manually:

- <https://gchq.github.io/CyberChef/>
- Decompiler: <https://godbolt.org/>

Info inside an APK

- Permissions: There are permissions in AndroidManifest.XML that can tell how an app intends to communicate with resources
- Code and anti-reversing techniques: Some apps might be using obfuscation that shall be a signal of the need for deeper inspection
- API Calls and Dependencies: Analysis can identify vulnerable API calls or outdated dependencies that may have known security risks
- Data Storage: Examine how the app handles sensitive data or how the hardcoded credentials or unencrypted storage methods are used

Reverse engineer an APK

Apktool

A tool for reverse engineering 3rd party, closed, binary Android apps.

It can decode resources to nearly original form and rebuild them after making some modifications.

<https://apktool.org/>

Apktool

```
$ apktool d myApplication.apk
I: Using Apktool 2.9.2 on myApplication.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: 1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Apktool

```
$ apktool b myApplication
I: Using Apktool 2.9.2 on myApplication
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
```

Apktool

Executing `apktool` gives all the options of the tool.

- `-f,--force` ➡ Force delete destination directory.
- `-o,--output`
➡ The name of folder that gets written. (default: apk.out)
- `-p,--frame-path`
➡ Use framework files located in
.
- `-r,--no-res` ➡ Do not decode resources.
- `-s,--no-src` ➡ Do not decode sources.

Apktool build

The option **b** builds the smali/dex code back to an .apk

Before you can run a rebuild application it needs to be signed first.

Signing APK

First use zipalign. This is a zip archive alignment tool that helps ensure that all uncompressed files in the archive are aligned relative to the start of the file.

Use zipalign to optimize your APK file before distributing it to end users.

More information: <https://developer.android.com/tools/zipalign>

Signing APK

Android requires that all APKs be digitally signed with a certificate before they are installed on a device or updated.

This can be done via apksigner. First generate keys in Android Studio and then use these certificates to sign the APK.

More information about signing here:

- <https://developer.android.com/tools/apksigner>
- <https://developer.android.com/studio/publish/app-signing#signing-manually>

Demonstration

Details demonstration

Build the apk

```
apktool b Demo4 -o RecompiledDemo4.apk
```

Aligns uncompressed data in APK

```
zipalign -p -f 4 RecompiledDemo4.apk RecompiledAlignedDemo4.apk
```

Details demonstration

Create a signing key

```
keytool -genkey -v -keystore keykoenk -alias keykoenk -keyalg RSA  
-keysize 2048 -validity 10000
```

Sign the apk with the signing key

```
apksigner sign --ks keykoenk --v1-signing-enabled true --v2-  
signing-enabled true RecompiledAlignedDemo4.apk
```

Install the new apk (delete the old one first)

```
adb install RecompiledAlignedDemo4.apk
```

Smali code

Smali code

When you understand how a developer builds something, it makes it much easier to understand how to reverse engineer it.

```
# direct methods
.method static constructor <clinit>()v
    .locals 1

    const/16 v0, 0x2004

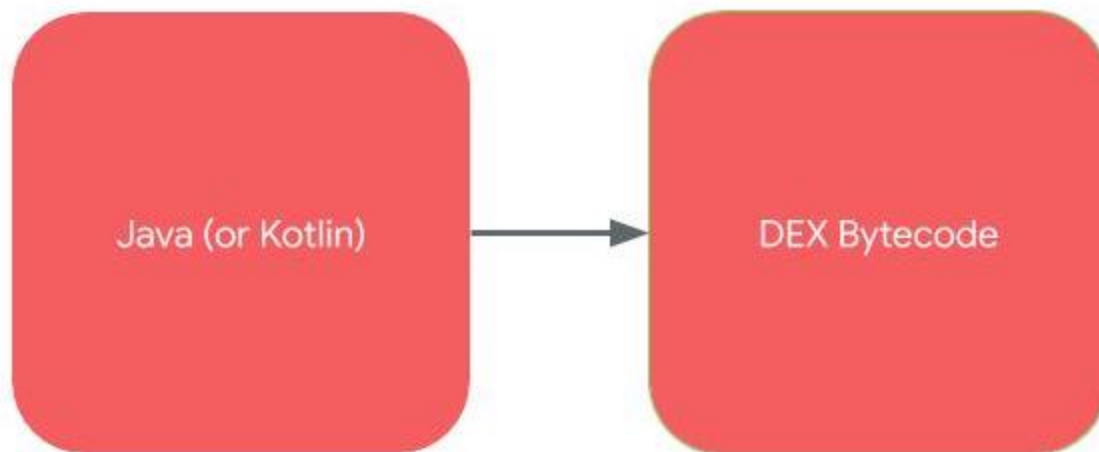
    new-array v0, v0, [B

    fill-array-data v0, :array_0
```

Smali code

Instead of the Java code being run in Java Virtual Machine (JVM) like desktop applications, in Android, the Java is compiled to the Dalvik Executable (DEX) bytecode format.

Developers



Smali code

Reverse Engineers



Smali code

Java code & Smali result

```
public static void printHelloWorld() {  
    System.out.println("Hello World")  
}
```

```
.method public static printHelloWorld()V  
    .registers 2  
    sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;  
    const-string v1, "Hello World"  
    invoke-virtual {v0,v1}, Ljava/io/PrintStream; ->println(Ljava/lang/String;)V  
    return-void  
.end method
```


Smali code

Smali represents Android's Dalvik bytecode in a readable text form, where each .smali file corresponds to a class.

- Registers (e.g., v0, v1) hold values like variables each method declares how many registers it uses (.locals or .registers directive)
- Basic instructions include const (load constants), move (copy values between registers), invoke-* (method calls), return-* (return from methods), and field access such as sget-object (static field get)
- Data types map to specific letters, e.g., I for int, L for class, Z for boolean

Smali code

Keywords in SMALI

- method
- registers
- sget-object
- const-string ➡ variables used
- invoke-virtual ➡ function call
- return-void(object) ➡ return value
- end method

Smali code

- line ➡ The number of lines in the source java file for the current code
- param ➡ Input parameters
- prologue ➡ Represents the starting tag executed within the function.
- end method ➡ End tag of a function
- new-instance ➡ Creates a new object
- invoke-direct ➡ Represents a direct call using the no-argument constructor

Smali code

- `iput-object vx,vy,field_id` ➡ Puts the object reference in vx into an instance field. The instance is referenced by vy.
- `iget-char vx,vy,field_id` ➡ Reads a char instance field into vx. The instance is referenced by vy.
- `move vx,vy` ➡ Moves the content of vy into vx. Both registers must be in the first 256 register range.
- `return vx` ➡ Return with vx return value
 - `return v0` ➡ returns with the value in v0
- `return-object` ➡ The return value from the public method

Smali code

The registers are indexed from v0 onwards.

There are registers p0, p1, etc., for all the parameters to the function. The class instance (this) gets p0, then p1 is the first argument. If the function is static, then p0 becomes the first parameter instead.

Smali code

- Method definitions start with `.method`, end with `.end method`
- Instruction formats typically include opcode and registers or constants, e.g.: `const/16 v0, 42` sets register v0 to 42

- ```
invoke-virtual {v0, v1}, Ljava/io/PrintStream;-
>println(Ljava/lang/String;)V
```

  
calls println on v0 passing v1`Labels like :cond\_0 are branching targets for conditional jumps.

# Smali code

---

Java code

```
if(flagx == 1)
 flagx = 2
else
 flagx = 3
```

# Smali code

---

## SMALI result

```
V1 = 0x1 = 1
{IF} if-ne (v0(flagx) == v1)
 v2 = 0x2
 v0 = v2
 goto: goto_0 = END
{ELSE} v2 = 0x3
 v0 = V2
{END} :goto_0
```



# Smali code

---

More information about SMALI code:

- <https://source.android.com/devices/tech/dalvik/dalvik-bytecode>
- [http://pallergabor.uw.hu/androidblog/dalvik\\_opcodes.html](http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html)

# Demonstration

# APKHunt

---

APKHunt is a comprehensive static code analysis tool for Android apps that is based on the OWASP MASVS framework.

# OWASP MASVS Static Analyzer

## APK Hunt Usage:

Options:

Examples:

Note:

**howest**  
hogeschool

# APKHunt

---

Scan coverage:

Covers most of the SAST (Static Application Security Testing) related test cases of the OWASP MASVS framework.

- Multiple APK scanning: Supports scanning multiple APK files
- Optimised scanning: Specific rules are designed to check for particular security sinks
- Low false-positive rate: Designed to pinpoint and highlight the exact location of potential vulnerabilities in the source code

# APKHunt

---

[\*] Reference:

- OWASP MASVS: MSTG-STORAGE-10 | CWE-316: Cleartext Storage of Sensitive Information in Memory
- [https://mobile-security.gitbook.io/masvs/security-requirements/0x07-v2-data\\_storage\\_and\\_privacy\\_requirements](https://mobile-security.gitbook.io/masvs/security-requirements/0x07-v2-data_storage_and_privacy_requirements)

==>> The possible Hard-coded Information...

```
/home/koenk/DemoDecompile_SAST/sources/androidx/fragment/app/FragmentManager.java
162: String key = "f" + i;
```

- With every result there is a reference to the MASVS with notes included

# APKHunt

---

==>> The Permissions...

```
/home/koenk/DemoDecompile_SAST/resources/AndroidManifest.xml
4: <uses-permission android:name="android.permission.INTERNET"/>
5: <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
6: <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
7: <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
8: <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
9: <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
...
17: <uses-permission android:name="android.permission.RECORD_AUDIO"/>
18: <uses-permission android:name="android.permission.CAMERA"/>
19: <uses-permission android:name="android.permission.READ_SMS"/>
20: <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
21: <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
22: <uses-permission android:name="android.permission.SET_WALLPAPER"/>
23: <uses-permission android:name="android.permission.READ_CALL_LOG"/>
24: <uses-permission android:name="android.permission.WRITE_CALL_LOG"/>
25: <uses-permission android:name="android.permission.WAKE_LOCK"/>
26: <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
27: <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

# APKHunt

---

Installation info: <https://github.com/Cyber-Buddy/APKHunt>



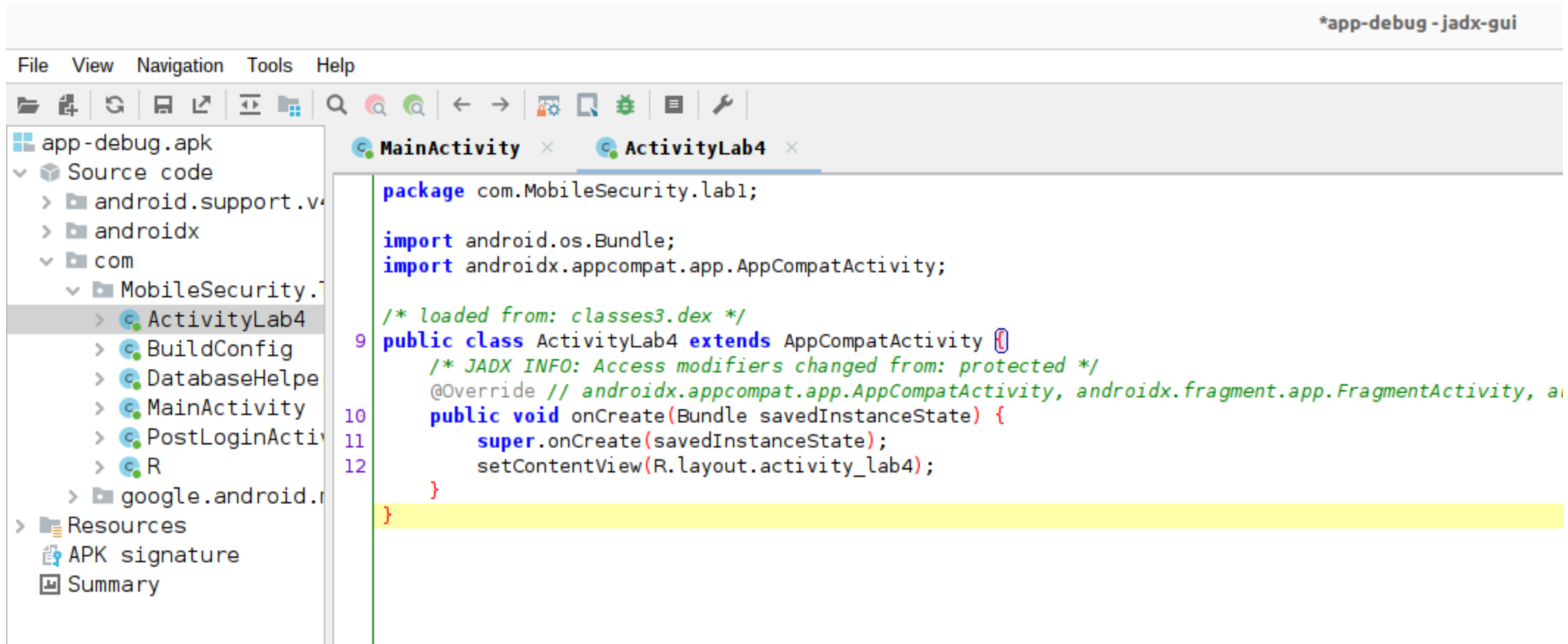
# JADX

---

Command line and GUI tools for producing Java source code from Android Dex and Apk files.

<https://github.com/skylot/jadx>

# JADX



The screenshot displays the JADX GUI interface. The title bar at the top right reads '\*app-debug - jadx-gui'. The menu bar includes 'File', 'View', 'Navigation', 'Tools', and 'Help'. Below the menu is a toolbar with various icons for file operations and navigation. On the left, a project tree shows the file structure: 'app-debug.apk' > 'Source code' > 'android.support.v4' > 'androidx' > 'com' > 'MobileSecurity.lab1' > 'ActivityLab4'. The 'ActivityLab4' file is selected. The main editor area shows the decompiled Java code for 'MainActivity'. The code is as follows:

```
package com.MobileSecurity.lab1;

import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;

/* loaded from: classes3.dex */
9 public class MainActivity extends AppCompatActivity {
 /* JADX INFO: Access modifiers changed from: protected */
 @Override // androidx.appcompat.app.AppCompatActivity, androidx.fragment.app.FragmentActivity, android.os.Bundle
10 public void onCreate(Bundle savedInstanceState) {
11 super.onCreate(savedInstanceState);
12 setContentView(R.layout.activity_lab4);
 }
}
```

# Bytecode viewer

---

Gui tool to decompile Android APK files to Java

<https://bytecodeviewer.com/>

# Demonstration

# Tooling

---

Notable other tools:

- Semgrep: <https://github.com/semgrep/semgrep>
- Androguard: <https://github.com/androguard/androguard>
- HCL AppScan: <https://www.hcl-software.com/appscan>
- SonarQube

**Lab time**

**See Lab4 on LEHO**