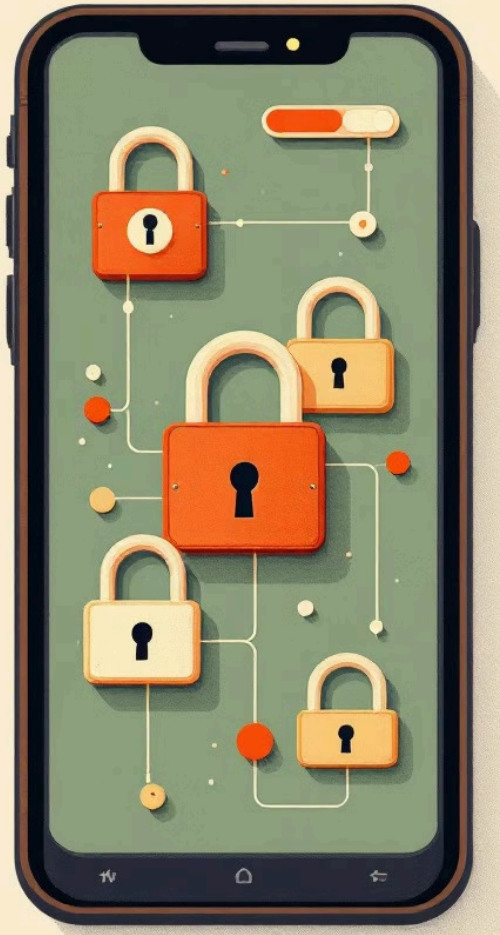
Android Application Penetration Testing

This workshop will explore techniques and methodologies for assessing the security posture of Android applications, from architectural analysis to hands-on exploitation of common vulnerabilities.

SK **by Sheshananda Reddy Kandula**



Whoami

@Sheshananda Reddy Kandula

Agenda

- Introductions
- Introduction to Android Application Structure
- Platform Security Controls that protect Android app
- Android Studio and Installing Emulator
- Building a Simple Android App with Android Studio
- Tools for pen testing Android App
- Damn Vulnerable Apps for learning
- Exploiting few vulnerabilities,
- Next Steps to take it further

Agenda

Topics Estimated Timeline

Start @ 1 PM EST

- Introductions
- Introduction to Android Application Structure
- Platform Security Controls that protect Android app
- Core Android App Components

2 PM EST

- Android Studio and Installing Emulator
- Building a Simple Android App with Android Studio
- Common vulnerabilities in Android app and Mobile OWASP Top 10

3 PM EST

- Static Code Analysis Techniques
- Dynamic Code Analysis Techniques
- Tools for pen testing Android App
- Damn Vulnerable Apps for learning
- Installing Burp Certificate on Emulator/Device for Proxy

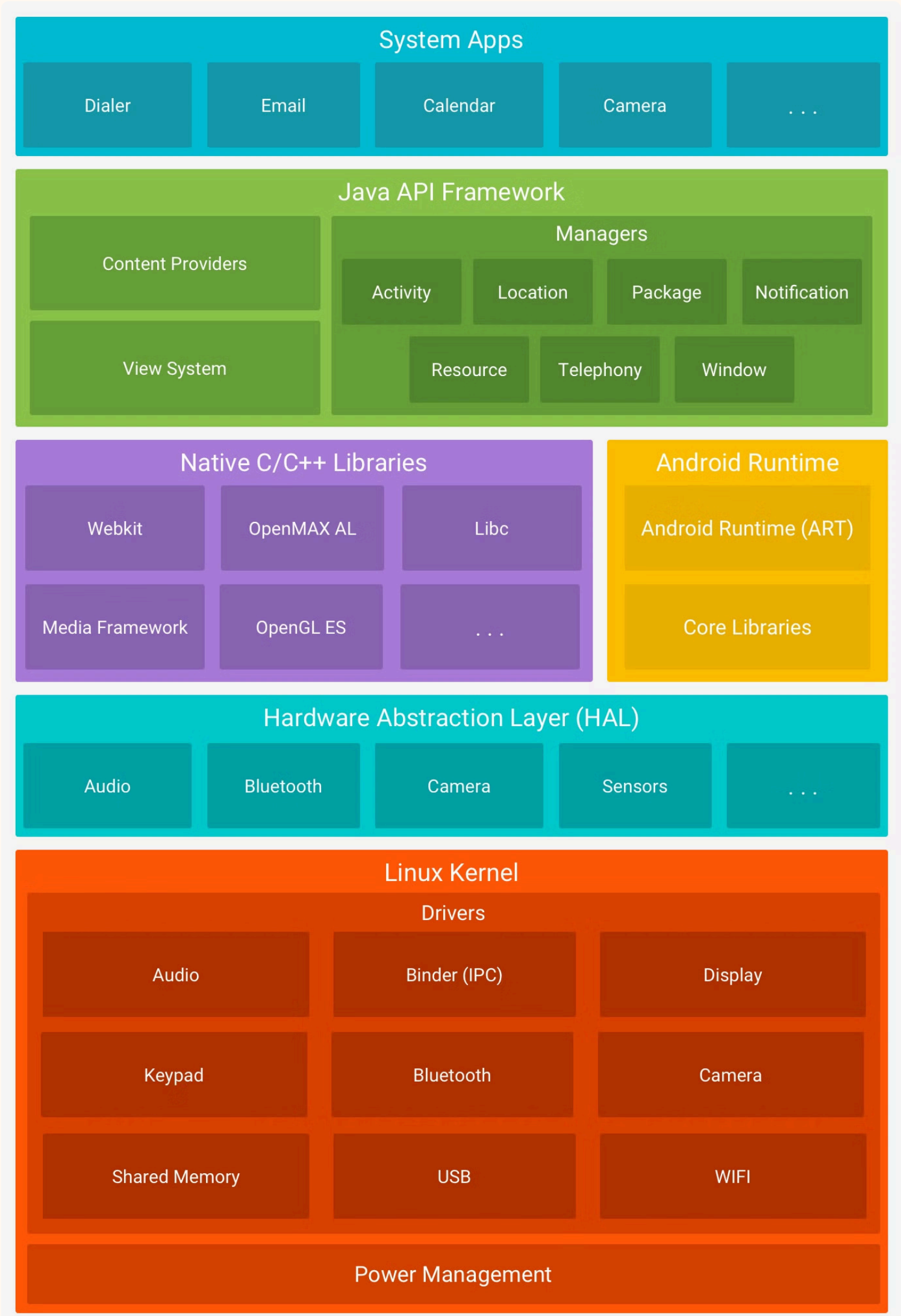
4 PM

- Exploiting few vulnerabilities,
- Next Steps to take it further

Introduction to Android Application Security

Android's open-source nature and diverse ecosystem introduce unique security challenges. Apps handle sensitive user data and often integrate with device hardware and services. **Secure design, coding practices, and rigorous testing are critical to mitigate risks.** Penetration testing helps identify and address vulnerabilities before malicious exploitation.

Android Platform Architecture



https://developer.android.com/guide/platform

Android Platform Security Controls

- **Application Sandbox**
  - Each app runs in an isolated environment, preventing unauthorized access to other apps' data.
- **Permissions System**
  - **Runtime Permissions** (Android 6.0+): User-granted permissions for sensitive actions.
- **Encryption & Storage Security**
  - **File-Based Encryption (FBE)**: Secures data at rest with hardware-backed encryption.
  - **KeyStore & TEE**: Protects cryptographic keys and sensitive operations.
- **Authentication Mechanisms**
  - **Biometrics & Multi-Factor**: Secure access using fingerprints, facial recognition, and traditional PIN/password.
- **Secure Boot & Verified Boot**
  - Validates each OS layer during boot to ensure integrity, with **rollback protection**.
- **Google Play Protect**
  - Scans apps for malware and enforces app security on the Play Store.
- **SELinux & Access Control**
  - Enforces strict security policies, containing apps within defined boundaries.

Android Application Architecture and Ecosystem

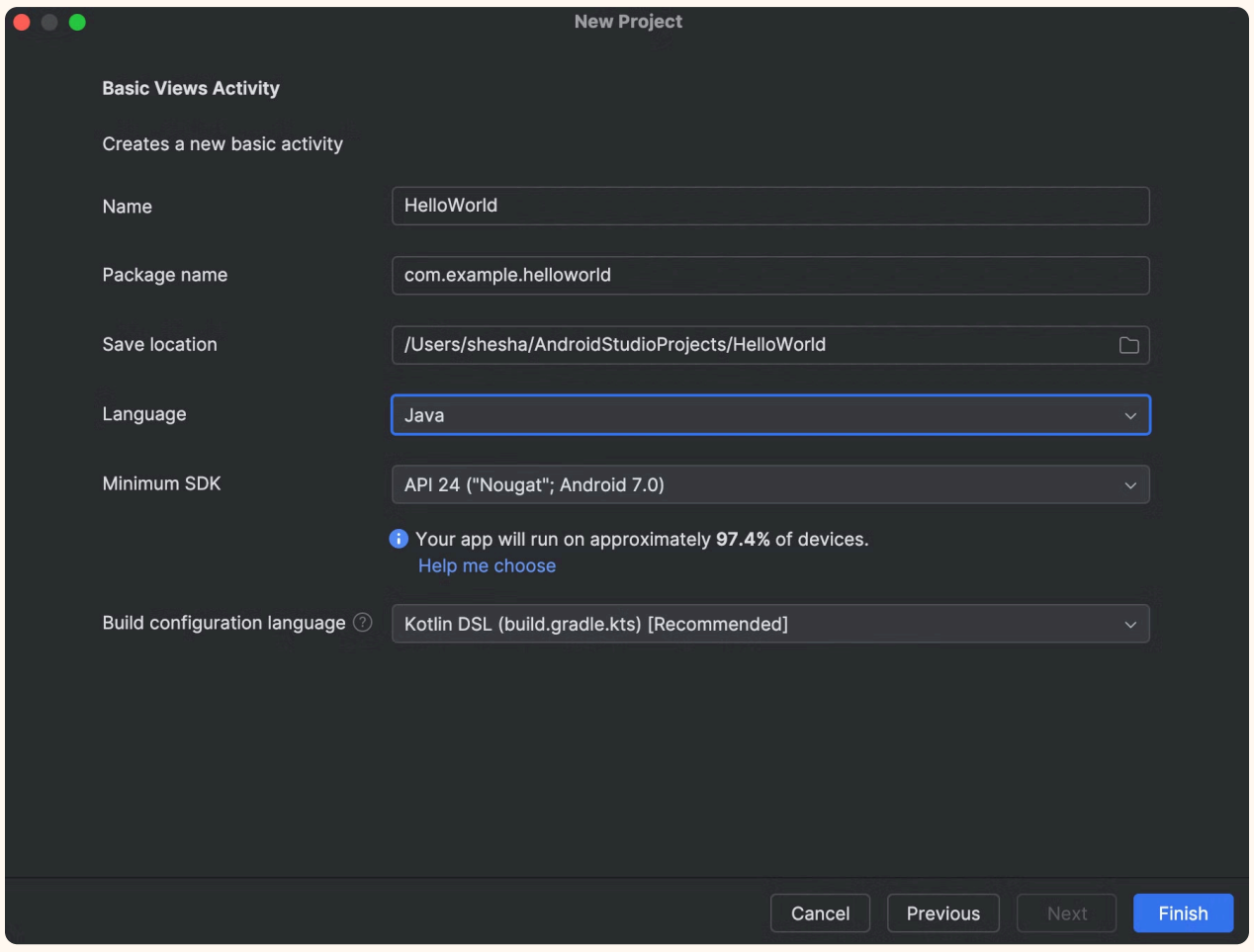
- **Android apps are built using a modular architecture with distinct components like Activities, Services, and Broadcast Receivers.**
- **The Android OS and its extensive API ecosystem enable rich functionality but also introduce potential security risks.**
- **The open Android app distribution model allows for a diverse range of apps, many of which may have vulnerabilities.**
- Understanding the Android platform's architecture and ecosystem is crucial for effective penetration testing.

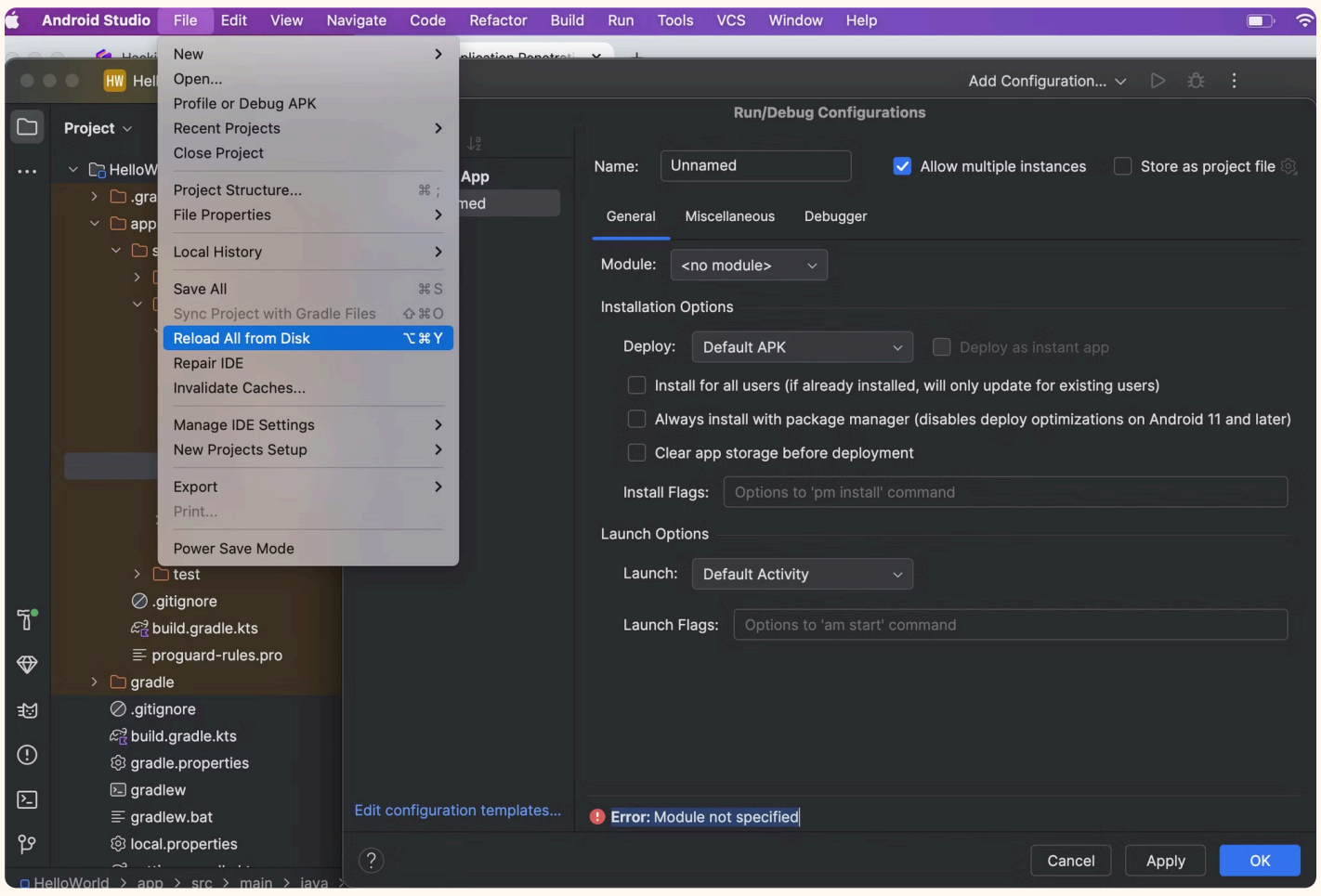
Core Android Application Components

1. **Activities**
   - **Purpose**: Manages the user interface and handles user interactions for a single screen.
   - **Example**: Login screen, profile page.
2. **Services**
   - **Purpose**: Runs long-running operations in the background without a UI.
   - **Example**: Playing music, fetching data, sending notifications.
3. **Broadcast Receivers**
   - **Purpose**: Responds to system-wide broadcast announcements (e.g., battery low, Wi-Fi connected).
   - **Example**: Listening for boot completion or network changes.
4. **Content Providers**
   - **Purpose**: Manages shared data between apps securely, allowing controlled access to databases.
   - **Example**: Contacts app sharing data with the messaging app.

Hello World Android Application

Let's Start with creation of Sample App.





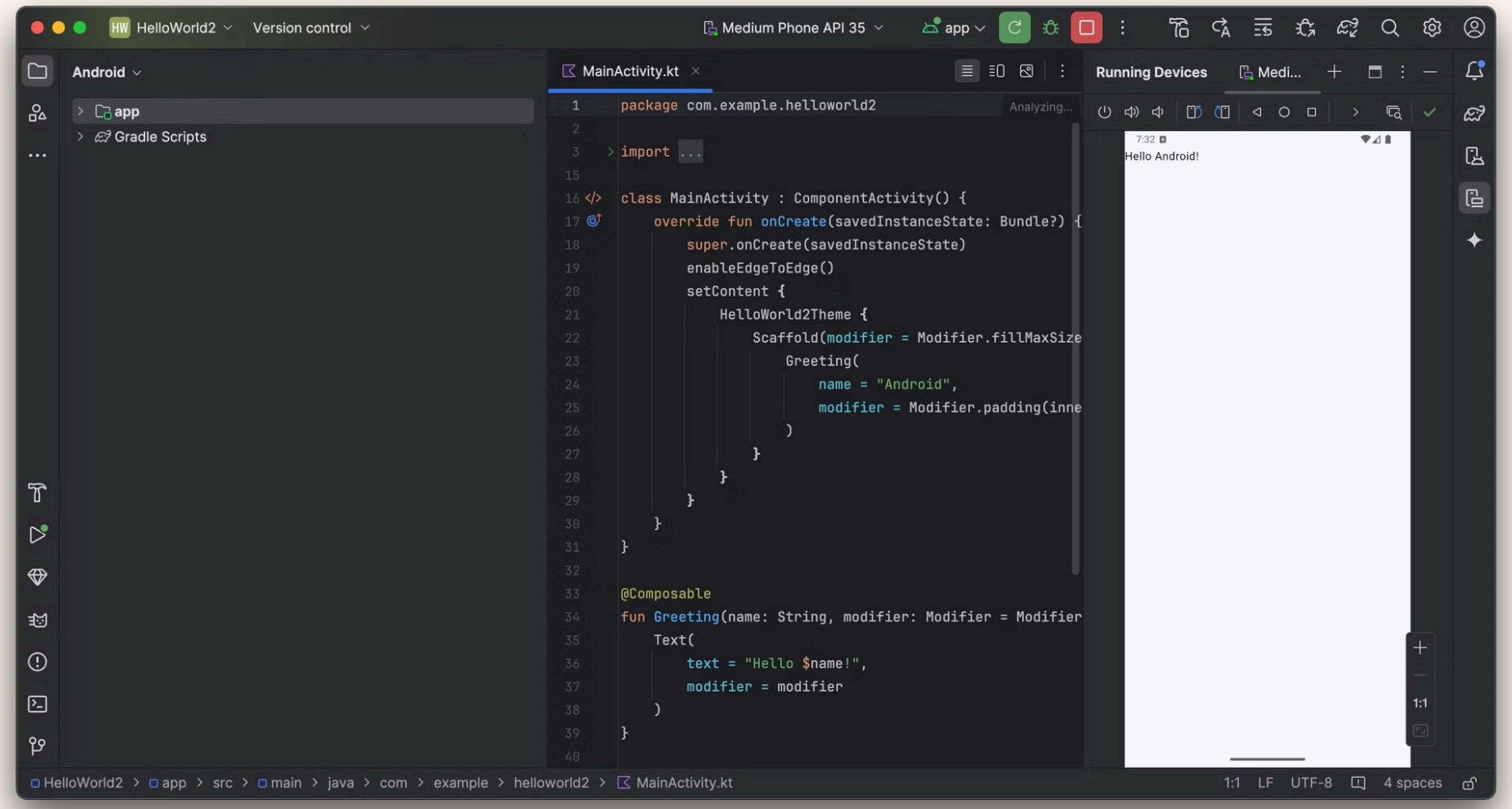
Developers

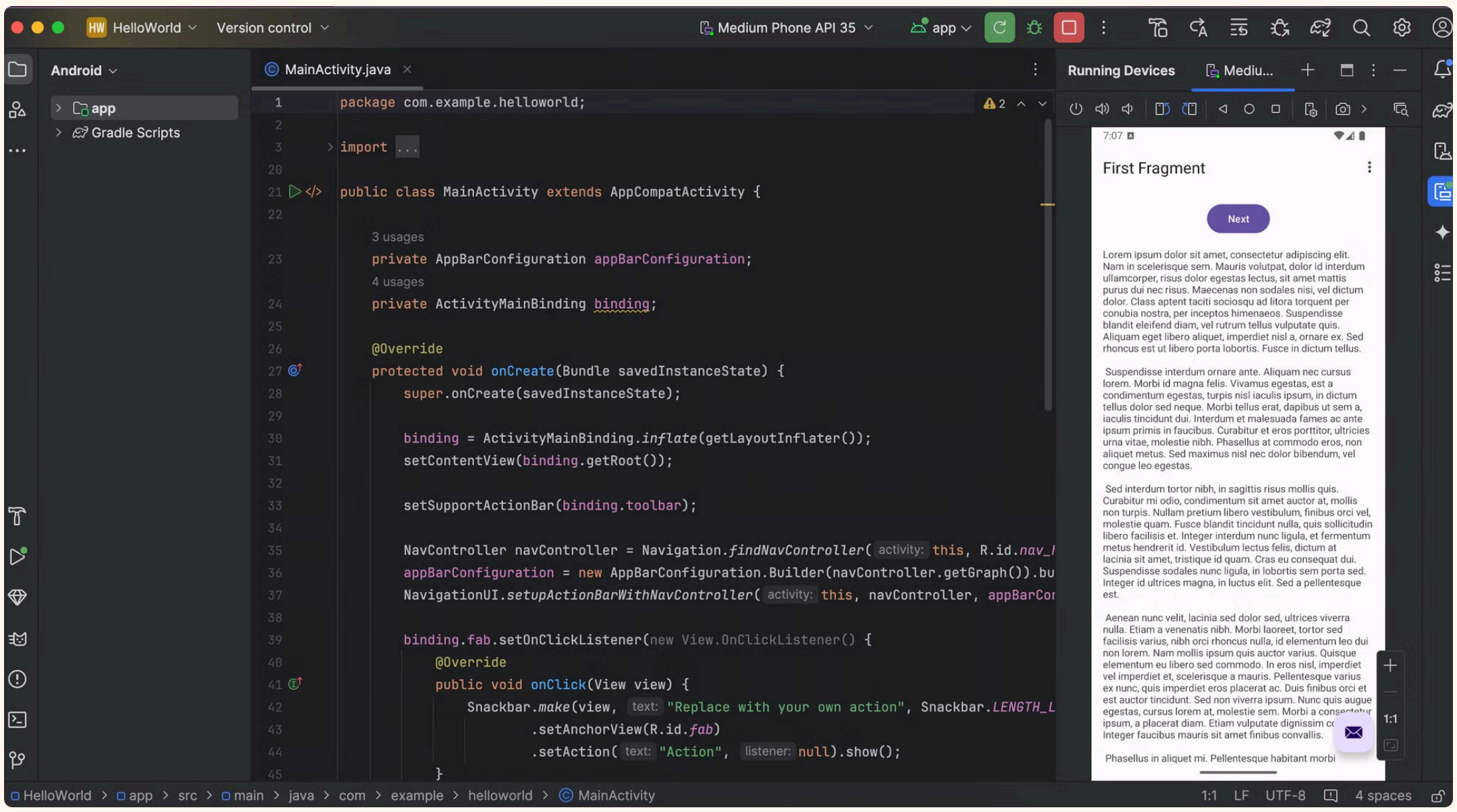
Android Developers

Download Android Studio & App Tools - Android Developers

Android Studio provides app builders with an integrated development environment (IDE) optimized for Android apps. Download Android Studio today.

First App


HelloWorld2
Version control
Medium Phone API 35
app
Android
app
Gradle Scripts
MainActivity.kt
Analyzing...
package com.example.helloworld2
import ...
class MainActivity : ComponentActivity() {
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
enableEdgeToEdge()
setContent {
HelloWorld2Theme {
Scaffold(modifier = Modifier.fillMaxSize
Greeting(
name = "Android",
modifier = Modifier.padding(inne
)
}
}
}
}
}
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier
Text(
text = "Hello $name!",
modifier = modifier
)
}
Running Devices
Medi...
7:32
Hello Android!
1:1
HelloWorld2 > app > src > main > java > com > example > helloworld2 > MainActivity.kt
1:1 LF UTF-8 4 spaces


HelloWorld
Version control
Medium Phone API 35
app
Android
app
Gradle Scripts
MainActivity.java
package com.example.helloworld;
import ...
public class MainActivity extends AppCompatActivity {
3 usages
private AppBarConfiguration appBarConfiguration;
4 usages
private ActivityMainBinding binding;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
binding = ActivityMainBinding.inflate(getLayoutInflater());
setContentView(binding.getRoot());
setSupportActionBar(binding.toolbar);
NavController navController = Navigation.findNavController(activity: this, R.id.nav_
appBarConfiguration = new AppBarConfiguration.Builder(navController.getGraph()).bu
NavigationUI.setupActionBarWithNavController(activity: this, navController, appBarCo
binding.fab.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
Snackbar.make(view, text: "Replace with your own action", Snackbar.LENGTH_L
.setAnchorView(R.id.fab)
.setAction(text: "Action", listener: null).show();
}
Running Devices
Mediu...
7:07
First Fragment
Next
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam in scelerisque sem. Mauris volutpat, dolor id interdum ullamcorper, risus dolor egestas lectus, sit amet mattis purus dui nec risus. Maecenas non sodales nisi, vel dictum dolor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Suspendisse blandit eleifend diam, vel rutrum tellus vulputate quis. Aliquam eget libero aliquet, imperdiet nisl a, ornare ex. Sed rhoncus est ut libero porta lobortis. Fusce in dictum tellus.
Suspendisse interdum ornare ante. Aliquam nec cursus lorem. Morbi id magna felis. Vivamus egestas, est a condimentum egestas, turpis nisl iaculis ipsum, in dictum tellus dolor sed neque. Morbi tellus erat, dapibus ut sem a, iaculis tincidunt dui. Interdum et malesuada fames ac ante ipsum primis in faucibus. Curabitur et eros porttitor, ultricies urna vitae, molestie nibh. Phasellus at commodo eros, non aliquet metus. Sed maximus nisl nec dolor bibendum, vel congue leo egestas.
Sed interdum tortor nibh, in sagittis risus mollis quis. Curabitur mi odio, condimentum sit amet auctor at, mollis non turpis. Nullam pretium libero vestibulum, finibus orci vel, molestie quam. Fusce blandit tincidunt nulla, quis sollicitudin libero facilisis et. Integer interdum nunc ligula, et fermentum metus hendrerit id. Vestibulum lectus felis, dictum at lacinia sit amet, tristique id quam. Cras eu consequat dui. Suspendisse sodales nunc ligula, in lobortis sem porta sed. Integer id ultrices magna, in luctus elit. Sed a pellentesque est.
Aenean nunc velit, lacinia sed dolor sed, ultrices viverra nulla. Etiam a venenatis nibh. Morbi laoreet, tortor sed facilisis varius, nibh orci rhoncus nulla, id elementum leo dui non lorem. Nam mollis ipsum quis auctor varius. Quisque elementum eu libero sed commodo. In eros nisl, imperdiet vel imperdiet et, scelerisque a mauris. Pellentesque varius ex nunc, quis imperdiet eros placerat ac. Duis finibus orci et est auctor tincidunt. Sed non viverra ipsum. Nunc quis augue egestas, cursus lorem at, molestie sem. Morbi a consectetur ipsum, a placerat diam. Etiam vulputate dignissim co
Integer faucibus mauris sit amet finibus convallis.
Phasellus in aliquet mi. Pellentesque habitant morbi
1:1
HelloWorld > app > src > main > java > com > example > helloworld > MainActivity
1:1 LF UTF-8 4 spaces

Installing ADB (Android Debugging Bridge)

Install the **homebrew** package manager

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```

Install adb

```
brew install android-platform-tools
```

Start using adb

```
adb devices
```

Using Android Studio Emulator with Root Access

The Android Studio Emulator has built-in support for root access, though not all system images allow rooting. Here's how to set it up:

1. **Open Android Studio**:
   - Open Android Studio, and go to **Tools > AVD Manager** (Android Virtual Device Manager).
2. **Create a New Virtual Device**:
   - Click on **Create Virtual Device** and select a device profile (e.g., Pixel 5).
3. **Select a System Image**:
   - Choose a system image that's compatible with rooting. **x86 or x86_64 images** (e.g., Android 11 or 12) often have better root support.
   - **Recommended:** Choose the system images labeled **Google APIs** rather than **Google Play** because Google Play images are usually locked down and cannot be rooted.
4. **Finish Setting Up the Emulator**:
   - Complete the setup and start the emulator.
5. **Verify Root Access**:
   - Once the emulator is running, open a terminal or use **Android Studio's Terminal**.
   - Run the following command to connect to the emulator:

Few commands:

- This command restarts the `adb` daemon with root privileges.
- Now, check for root access by running:
- If you see `root` as the output, the emulator has root access.

```
adb devices
adb root
adb shell
whoami
id
```

Conti...

Below is the expected behavior

```
(dev)  APKs git:(main)  adb devices
List of devices attached

(dev)  APKs git:(main)  adb devices
List of devices attached
emulator-5554      offline

(dev)  APKs git:(main)
(dev)  APKs git:(main)  adb devices
List of devices attached
emulator-5554      device

(dev)  APKs git:(main)  adb shell
emulator64_x86_64:/ $ su
emulator64_x86_64:/ # id
uid=0(root) gid=0(root)
groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),1078(ext_data_rw),1079(ext_obb_rw),3001(net_bt
_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid),3012(readtracefs) context=u:r:su:s0
emulator64_x86_64:/ # ls /data/data/
android                                  com.android.providers.contacts
android.auto_generated_rro_product__     com.android.providers.downloads
android.auto_generated_rro_vendor__      com.android.providers.downloads.ui
android.slim.overlay                     com.android.providers.media
com.android.backupconfirm                com.android.providers.partnerbookmarks
com.android.bips                         com.android.providers.settings
com.android.bluetooth                    com.android.providers.settings.auto_generated_rro_vendor__
com.android.bluetoothmidiservice         com.android.providers.telephony
com.android.bookmarkprovider             com.android.providers.userdictionary
com.android.calllogbackup                com.android.proxyhandler
com.android.cameraextensions             com.android.remoteprovisioner
com.android.carrierconfig                com.android.se
```

ADB - Android Debug Bridge

Android Developers

Android Debug Bridge (adb) | Android Studio | Android Developers

Find out about the Android Debug Bridge, a versatile command-line tool that lets you communicate with a device.

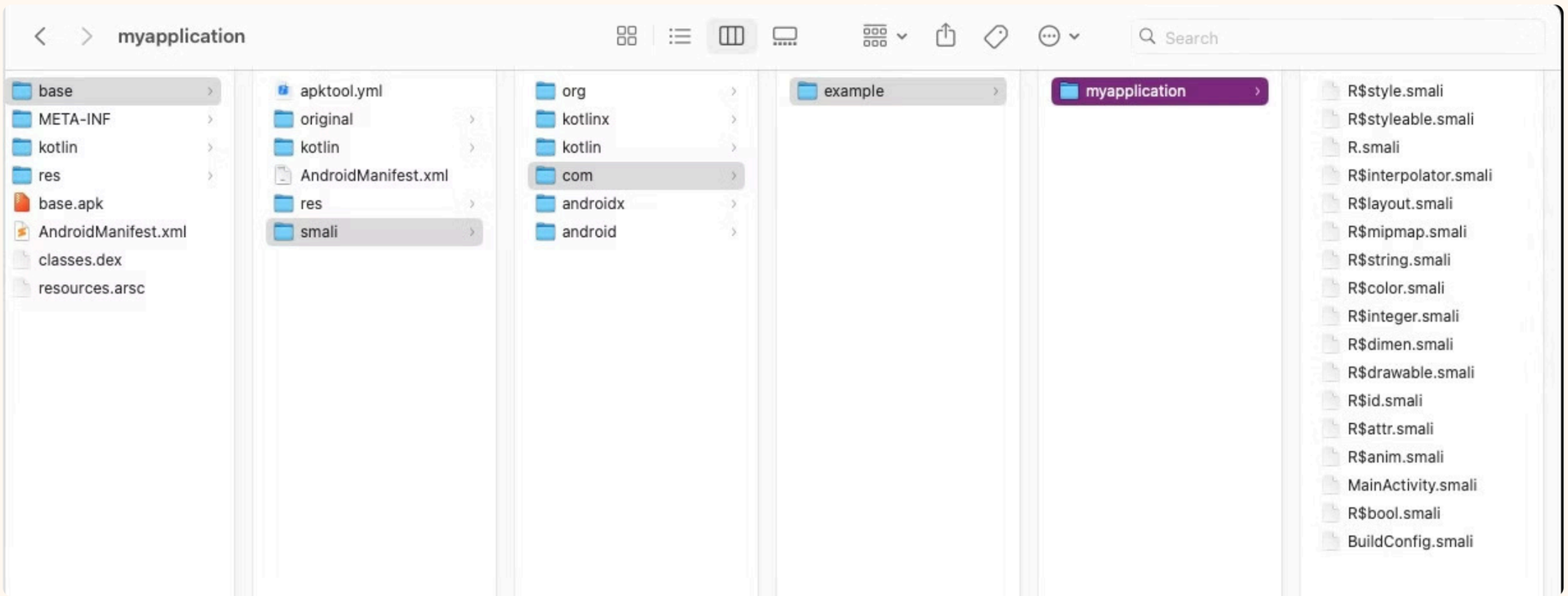
```
#basic commands
adb devices
adb shell
adb push ./file /sdcard/
adb pull /sdcard/file .
adb install file.apk

# Looking for the application and it's path
adb shell pm list packages
adb shell pm list packages -3
adb shell dumpsys package <package-name>
adb shell pm path <package-name>

# logcat
adb logcat "MainActivity:V *:S"
adb logcat -v "brief"
```

Helpful commands

```
adb exec-out screencap -p > screenshot.png
adb shell screencap /sdcard/screencap.png
adb shell screenrecord /sdcard/screenrecord.mp4
```

Android Application File Structure

```
staff     2296 Jan  1  1981 AndroidManifest.xml
staff      960 Nov  1 14:28 META-INF
staff  2198956 Nov  1 14:27 base.zip
staff  4216572 Jan  1  1981 classes.dex
staff      288 Nov  1 14:28 kotlin
staff      928 Nov  1 14:28 res
staff   250636 Jan  1  1981 resources.arsc
```

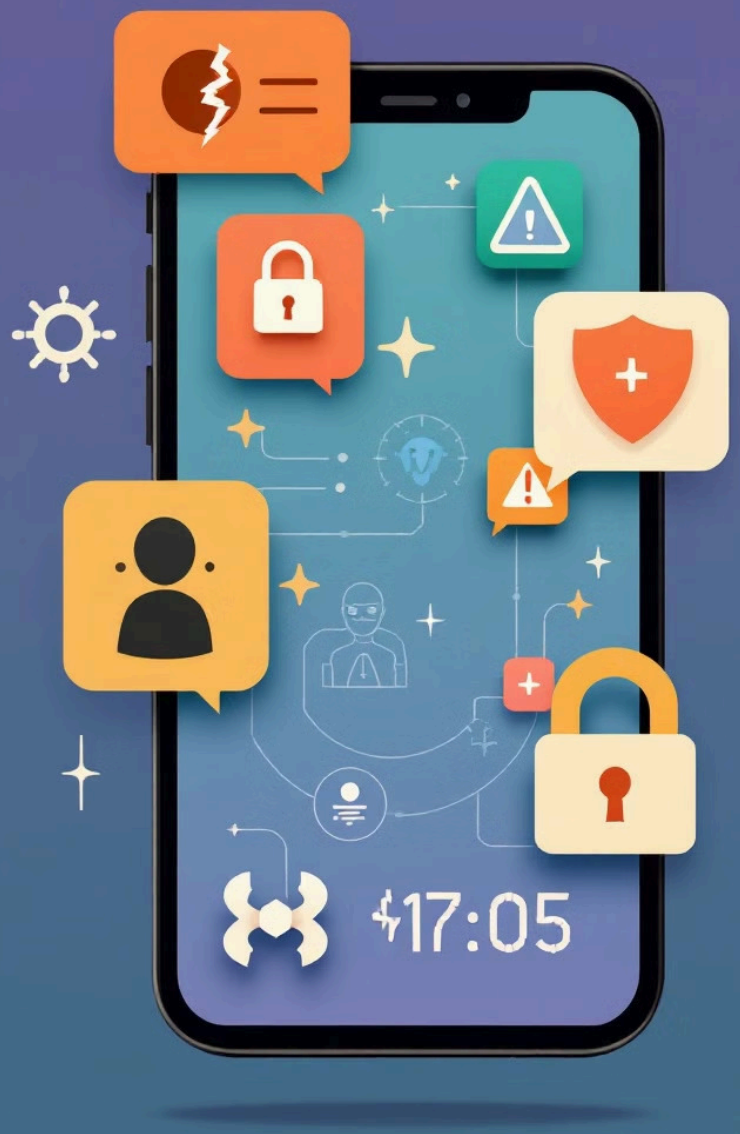


Android Application File Structure

```
→  Android-InsecureBankv2 git:(master) file InsecureBankv2.apk
InsecureBankv2.apk: Java archive data (JAR)
```

```
 extracting: resources.arsc
  inflating: classes.dex
  inflating: META-INF/MANIFEST.MF
  inflating: META-INF/CERT.SF
  inflating: META-INF/CERT.RSA
→  Android-InsecureBankv2 git:(master) ✗
```

```
staff     2296 Jan  1  1981 AndroidManifest.xml
staff      960 Nov  1 14:28 META-INF
staff  2198956 Nov  1 14:27 base.zip
staff  4216572 Jan  1  1981 classes.dex
staff      288 Nov  1 14:28 kotlin
staff      928 Nov  1 14:28 res
staff   250636 Jan  1  1981 resources.arsc
```



Common Android Security Vulnerabilities

1 **Insecure Data Storage**

Apps may store sensitive data like credentials and PII insecurely on the device.

2 **Authentication and Authorization Flaws**

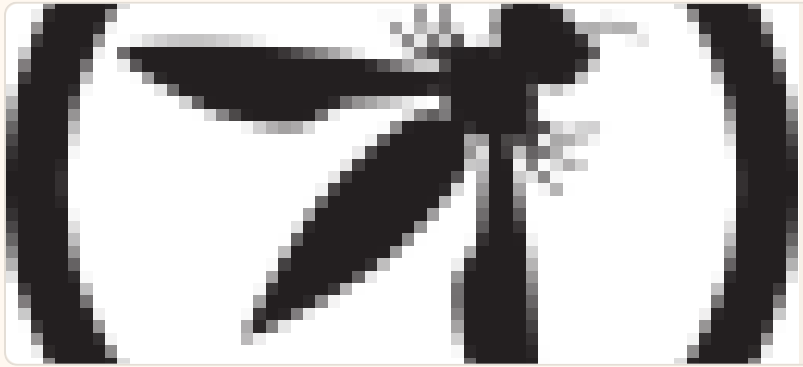
Weak or missing authentication mechanisms can allow unauthorized access.

3 **Vulnerable Inter-component Communication**

Improper handling of intents and other IPC channels can lead to exploits.

Mobile OWASP Top 10 Risks

- M1: Improper Credential Usage
- M2: Inadequate Supply Chain Security
- M3: Insecure Authentication/Authorization
- M4: Insufficient Input/Output Validation
- M5: Insecure Communication
- M6: Inadequate Privacy Controls
- M7: Insufficient Binary Protections
- M8: Security Misconfiguration
- M9: Insecure Data Storage
- M10: Insufficient Cryptography



owasp.org

OWASP Mobile Top 10 | OWASP Foundation

OWASP Mobile Top 10 on the main website for The OWASP Foundation. OWASP is a nonprofit foundation that works to improve the security of software.

Static Code Analysis Techniques

Automated Source Code Scanners

Automated source code scanners identify security anti-patterns, implementation errors, and compliance issues.

Manual Code Review

Manual code review by security experts uncovers subtle vulnerabilities and design flaws.

Decompilation Tools

Decompilation tools extract Java/Kotlin source code from Android APKs for in-depth analysis.

Taint Analysis

Taint analysis and data flow tracking detect improper handling of sensitive data.

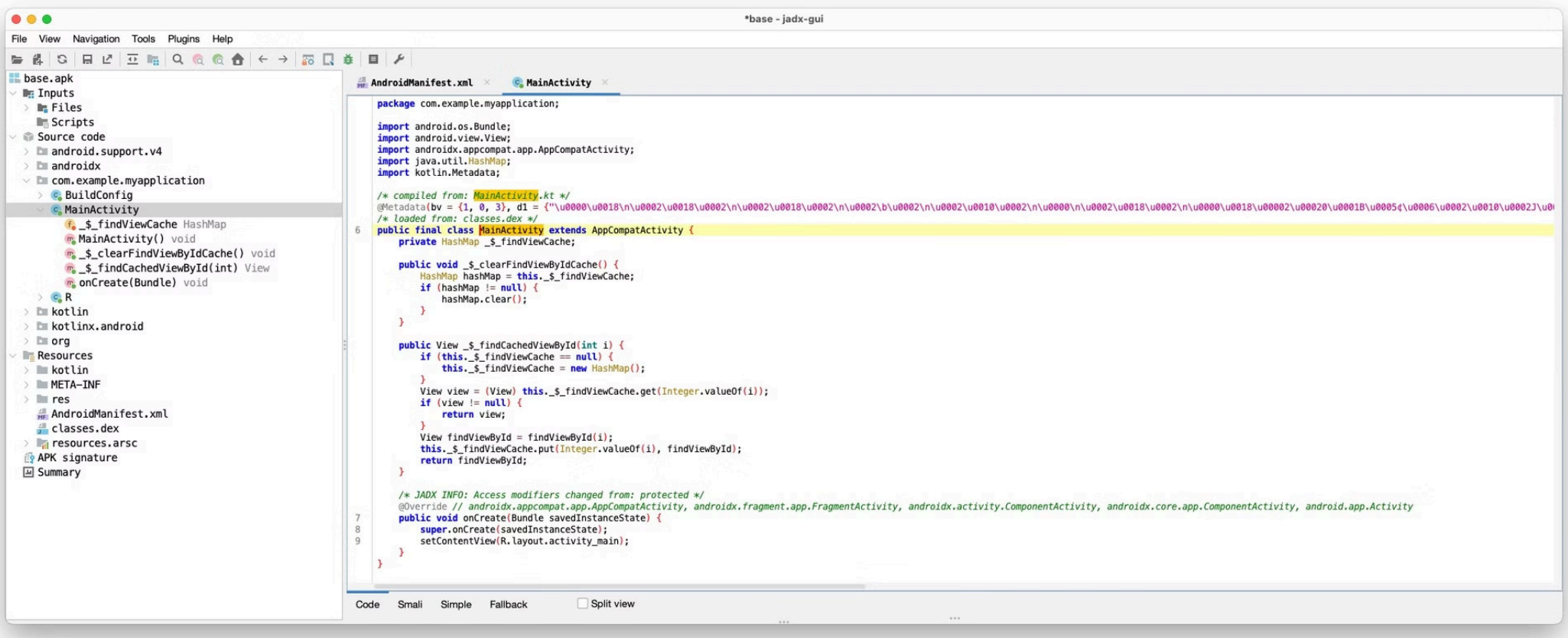
Static Code Analysis Tools

APK tool **https://apktool.org/**

Jadx-gui **https://github.com/skylot/jadx**

semgrep **https://github.com/mindedsecurity/semgrep-rules-android-security**

```
    tmp apktool d base.apk
I: Using Apktool 2.10.0 on base.apk with 12 thread(s).
I: Baksmaling classes.dex...
I: Loading resource table...
I: Decoding file-resources...
I: Loading resource table from file: /Users/dhanvi/Library/apktool/framework/1.apk
I: Decoding values */* XMLs...
I: Decoding AndroidManifest.xml with resources...
I: Regular manifest package...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```



Dynamic Analysis and Runtime Instrumentation

Runtime Analysis Tools

Runtime analysis tools like Frida and Xposed enable hooking into app execution for in-depth security evaluation.

1

2

Dynamic Instrumentation

Dynamic instrumentation techniques can monitor app behavior, manipulate inputs, and detect runtime vulnerabilities in a live environment.

Dynamic Tools

Frida : **https://frida.re/docs/android/**

```
(dev) → Android-InsecureBankv2 git:(master) ✗ frida-ps -Uai
  PID  Name                                  Identifier
-----  ------------------------------------  ------------------------------------
 8112  Chrome                                com.android.chrome
 7374  Facebook                              com.facebook.katana
12634  Google                                com.google.android.googlequicksearchbox
31120  Google Play Store                     com.android.vending
 6904  Maps                                  com.google.android.apps.maps
24033  Messages                              com.google.android.apps.messaging
 7175  Photos                                com.google.android.apps.photos
28195  Settings                              com.android.settings
 7895  YouTube                               com.google.android.youtube
```

Objection : **https://github.com/sensepost/objection**

pip3 install objection
frida-ps -Uai
objection --gadget asvid.github.io.fridaapp explore

```
(dev) → Android-InsecureBankv2 git:(master) ✗ objection
zsh: /usr/local/bin/objection: bad interpreter: /usr/local/opt/python/bin/python3.7: no such file or directory
Checking for a newer version of objection...
Usage: objection [OPTIONS] COMMAND [ARGS]...

       _   _           _   _
 ___| |_|_|___ ___| |_|_|___ ___
| . | . | | -_|  _|  _| | . |   |
|___|___| |___|___|_| |_|___|_|_|
      |___|(object)inject(ion)

      Runtime Mobile Exploration
         by: @leonjza from @sensepost

  By default, communications will happen over USB, unless the --network option
  is provided.

Options:
  -N, --network           Connect using a network connection instead of USB.
  -h, --host TEXT         [default: 127.0.0.1]
  -p, --port INTEGER      [default: 27042]
  -ah, --api-host TEXT    [default: 127.0.0.1]
  -ap, --api-port INTEGER [default: 8888]
  -g, --gadget TEXT       Name of the Frida Gadget/Process to connect to.
                          [default: Gadget]
  -S, --serial TEXT       A device serial to connect to.
  -d, --debug             Enable debug mode with verbose output. (Includes
                          agent source map in stack traces)
  --help                  Show this message and exit.

Commands:
  api          Start the objection API server in headless mode.
  device-type  Get information about an attached device.
  explore      Start the objection exploration REPL.
  patchapk     Patch an APK with the frida-gadget.so.
  patchipa     Patch an IPA with the FridaGadget dylib.
  run          Run a single objection command.
  signapk      Zipalign and sign an APK with the objection key.
  version      Prints the current version and exists.
```

Ref: **https://book.hacktricks.xyz/mobile-pentesting/android-app-pentesting/frida-tutorial/objection-tutorial**

Testing for Insecure Data Storage

1

Identify Sensitive Data

Identify app components that handle sensitive data like credentials, PII, and financial information.

2

Examine Data Storage

Examine where and how this data is stored on the device, including the Android file system and SQLite databases.

3

Assess Security Measures

Assess the use of encryption, access controls, and other security measures to protect data at rest.

Damn Vulnerable Apps

AndroGoat: **https://github.com/satishpatnayak/AndroGoat**

HexTree AttackSurface APK: **https://app.hextree.io/courses/intent-threat-surface/intents-and-activities/practice-startactivity**

```
(dev) →  APKs git:(main) ✗ adb shell
barbet:/ $ su
barbet:/ # cd /data/data/owasp.sat.agoat/
barbet:/data/data/owasp.sat.agoat # ls -l
total 9
drwxrws--x 2 u0_a378 u0_a378_cache 3452 2024-11-03 23:56 cache
drwxrws--x 2 u0_a378 u0_a378_cache 3452 2024-11-03 23:56 code_cache
drwxrwx--x 2 u0_a378 u0_a378       3452 2024-11-03 23:57 shared_prefs
barbet:/data/data/owasp.sat.agoat # more shared_prefs/users.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="password">123456</string>
    <string name="username">test</string>
</map>
barbet:/data/data/owasp.sat.agoat #
```

```
barbet:/data/data/owasp.sat.agoat # ls databases/
aGoat  aGoat-journal
barbet:/data/data/owasp.sat.agoat # cd databases/
barbet:/data/data/owasp.sat.agoat/databases # strings aG
aGoat          aGoat-journal
barbet:/data/data/owasp.sat.agoat/databases # strings aGoat
SQLite format 3
Ytablesqlite_sequencesqlite_sequence
CREATE TABLE sqlite_sequence(name,seq)}
Ytableusersusers
CREATE TABLE users (ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, username VARCHAR, password VARCHAR)W
ctableandroid_metadataandroid_metadata
CREATE TABLE android_metadata (locale TEXT)
en_US
test265321
        users
barbet:/data/data/owasp.sat.agoat/databases #
```

Identifying and Exploiting Authentication Flaws

Assess Authentication Controls

Assess authentication controls like password policies, multi-factor options, and account lockout mechanisms.

Bypass Authentication

Attempt to bypass, bruteforce, or otherwise circumvent authentication to gain unauthorized access to sensitive app functionality.

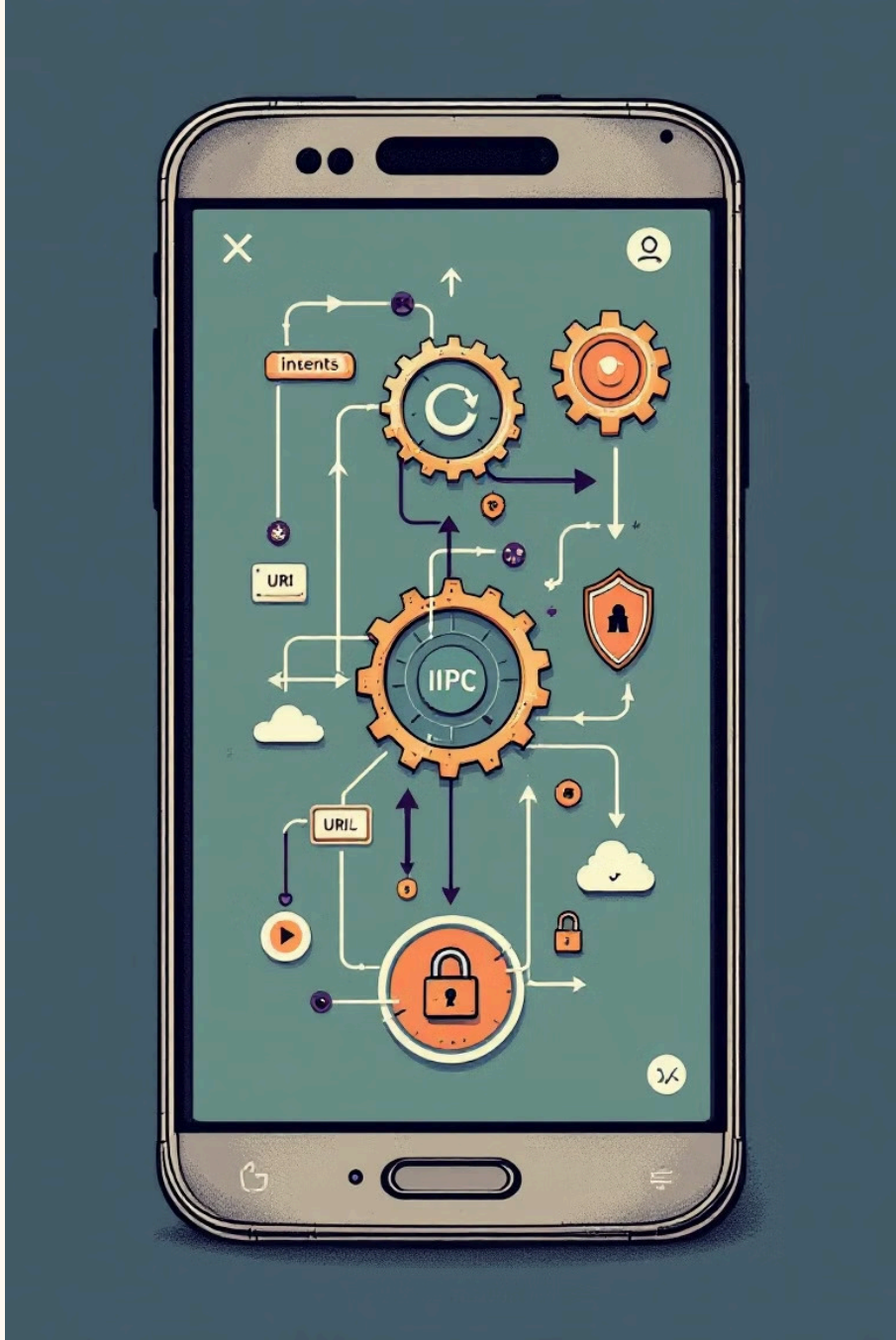
AndroGoat

Verify PIN Functionality to download the invoice

Attacking Android Inter-Component Communication

Android's open IPC model allows apps to share data and functionality, but can also enable unauthorized access if not properly secured.

Identify and analyze intent handling, URIs, and other IPC mechanisms for vulnerabilities that could lead to privilege escalation or data leakage.



AttackSurface APK

Intents attacks

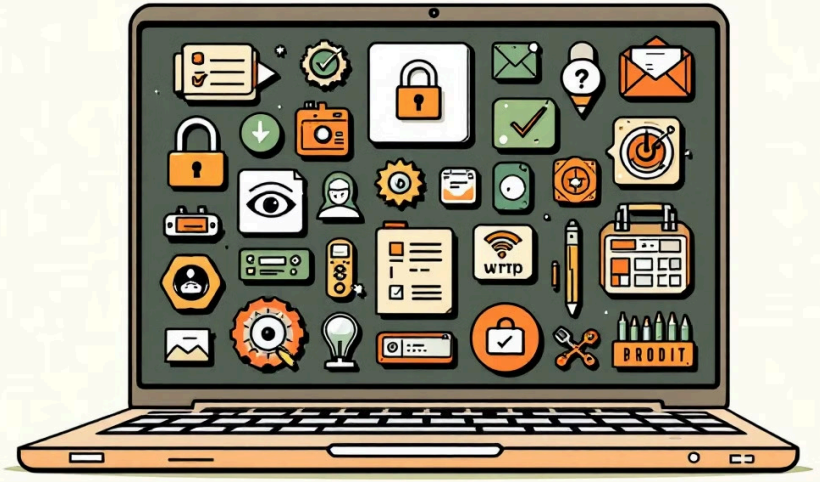
Flag 8 : Activity checking function name

Flag 10: Implicit Intent

MyDevices Beta

Automated Android Pentesting Tools and Frameworks

- **Frida enables dynamic instrumentation for runtime analysis and exploit development.**
- **MobSF is an open-source, all-in-one mobile app pentesting framework.**
- **Drozer is a comprehensive security assessment and exploitation framework for Android.**
- **Apktool allows for reverse-engineering, modifying, and rebuilding Android apps.**



MobSF Framework and Demo

MobSF: **https://github.com/MobSF/Mobile-Security-Framework-MobSF**

```
docker pull opensecurity/mobile-security-framework-mobsf:latest

docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest
```

Thank You!!


Sheshananda Reddy Kandula
Sr Security Engineer at Adobe | AppSec |
Product Securtiy | OSWE | OSCP | CISSP