Mobile

JOÃO PAULO BARRACA

Mobile landscape

Includes a wide a range of devices with low power characteristics

- Although we may be talking about an 8 core, +2GHz CPU
  - So… lots of potential computational power, which cannot be fully exploited due to battery limitations/power envelope

Smartphones: becoming the primary gateway through which users interact

- Dominated by two tech stacks: Android and iOS
- Supported application stores providing an easy access for app/content distribution
  - Application store acts and single point of control and can audit applications or enforce rules
- Devices are becoming increasingly secure and already enable 2FA, smart payments, …
  - Backed by hardware enclaves/trusted execution environments, secure encrypted storage, locked bootloaders,

Mobile landscape

Same tech stack is reused for other platforms… (mostly android)

- Smart TVs
- Car infotainment
- Home appliances
- Smart houses

Current data points towards more than 8.6 billion devices

- This is already above the number of people on earth

Anatomy of a mobile device (Hardware)

Modem: handles communications

- Closed source, provides ports to main CPU

SoC: main system including applicational CPU

- Runs kernel plus user applications
- May include a Trusted Execution Environment
  - TEE may be external

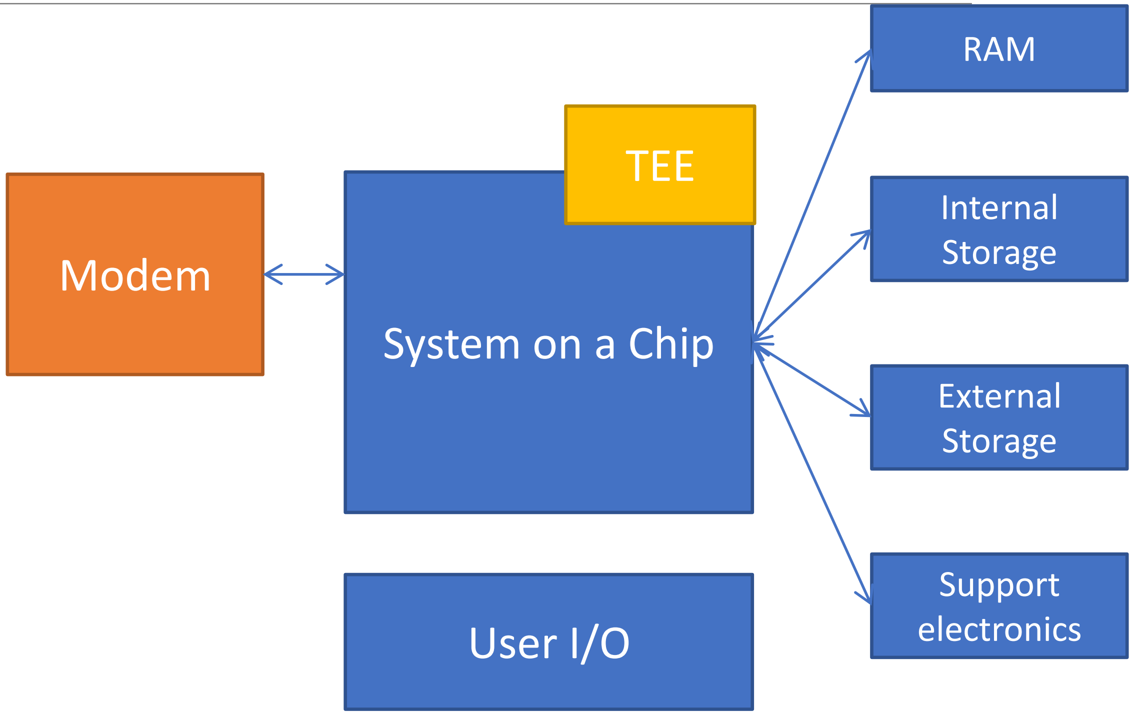
Internal Storage: NAND flash on device

- Soldered
- Typically encrypted in more recent models

External Storage: SD Card (optional)

- Upgradable by users
- Typically, not encrypted

User I/O touch screen + buttons + biometric



Anatomy of a mobile device (Software)

BootROM
- Read only code to boot device

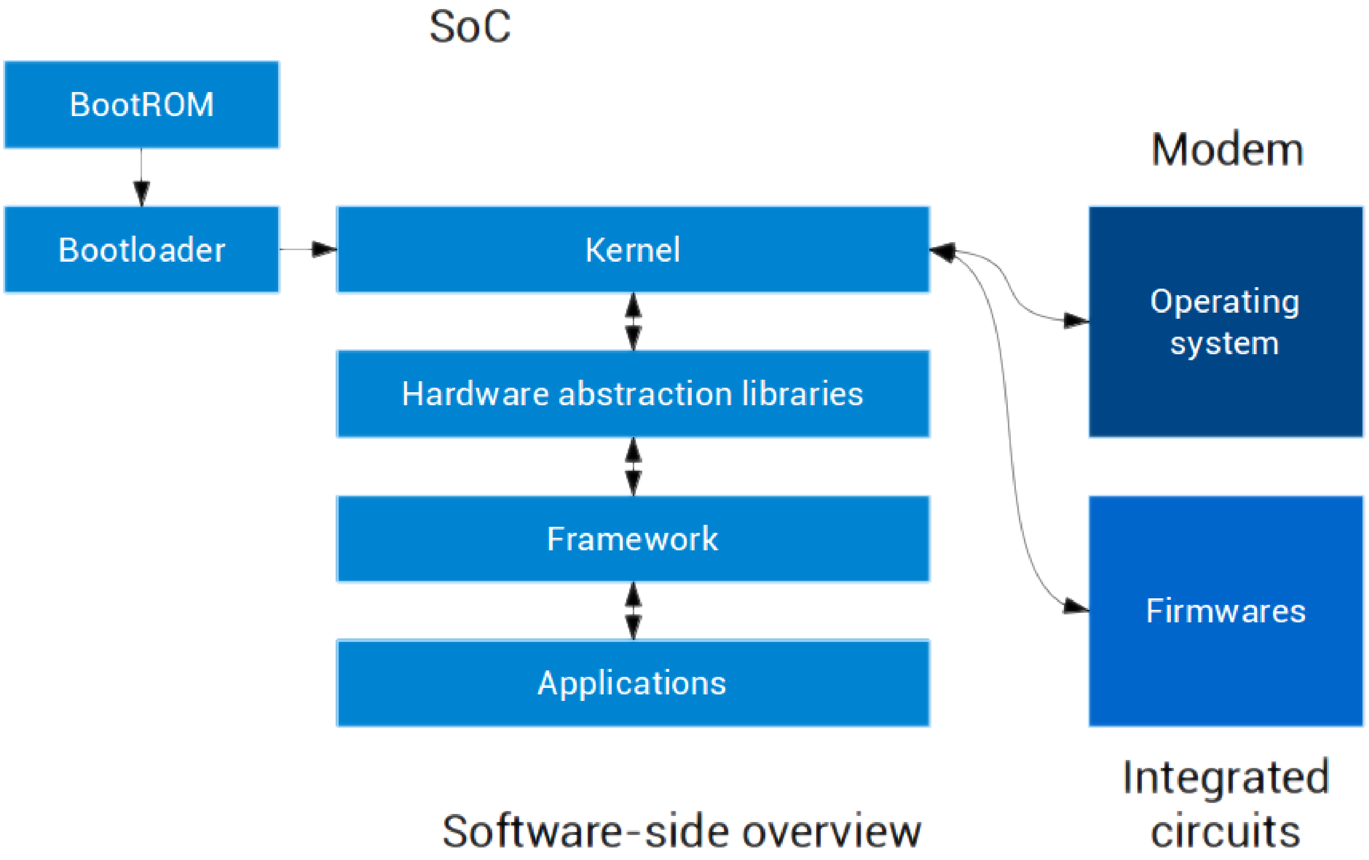
Bootloader
- Prepares the loading of a kernel
- May be locked: validates kernel auth

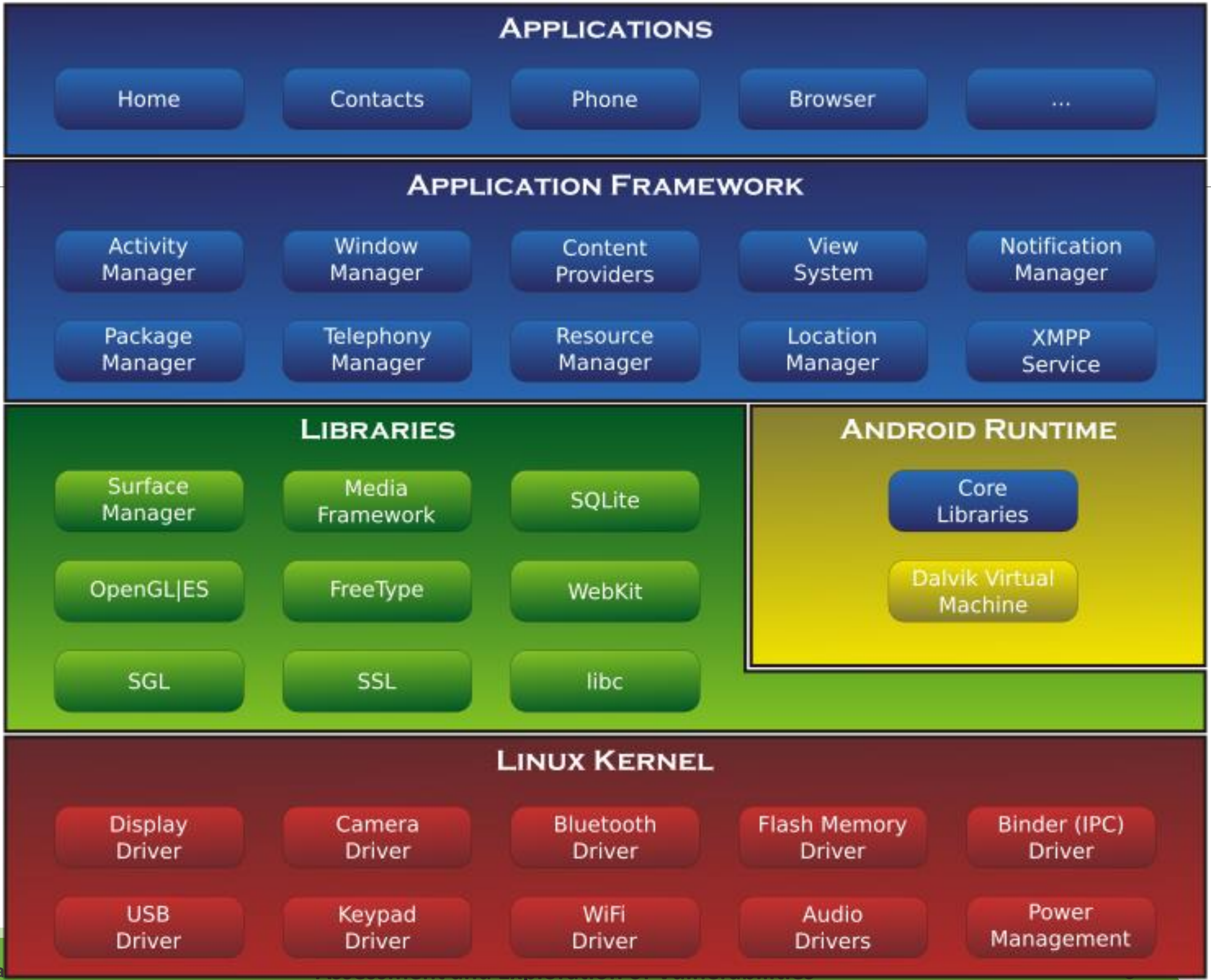
Kernel
- iOS/Linux/Windows kernel

HAL
- Provide access to hardware resources

Framework
- Set of classes through which applications interact

Application
- Software packages provides by multiple parties and users





Applications

Home
Contacts
Phone
Browser
...

Application Framework

Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
XMPP Service

Libraries

Surface Manager
Media Framework
SQLite
OpenGL|ES
FreeType
WebKit
SGL
SSL
libc

Android Runtime

Core Libraries
Dalvik Virtual Machine

Linux Kernel

Display Driver
Camera Driver
Bluetooth Driver
Flash Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Android Applications

Components deriving from primitive framework classes

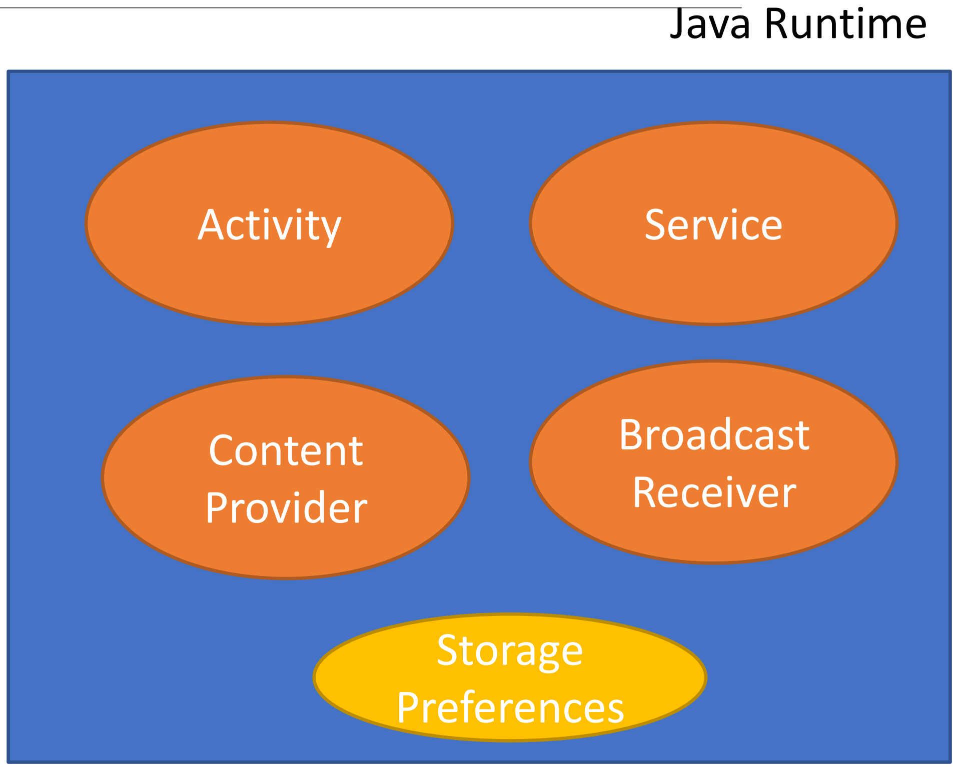
- **Activity**: a single, focused thing that the user can do
  - will usually take the whole screen
- **Service**: a component doing something or providing functionality
  - without UI presence
- **Broadcast Receiver**: a receiver of intents to handle events and IPC
- **Content Provider**: encapsulate data and provide it to applications

Assumes an asynchronous, non persistent model

- Applications can be stopped/paused/started/resumed at any time
- Intents are used as an important IPC to dispatch messages across components

All this is represented as Java/Kotlin classes

- Inherited by applications



Trusted Execution Environment (TEE)

An isolated environment that runs in parallel with the operating system

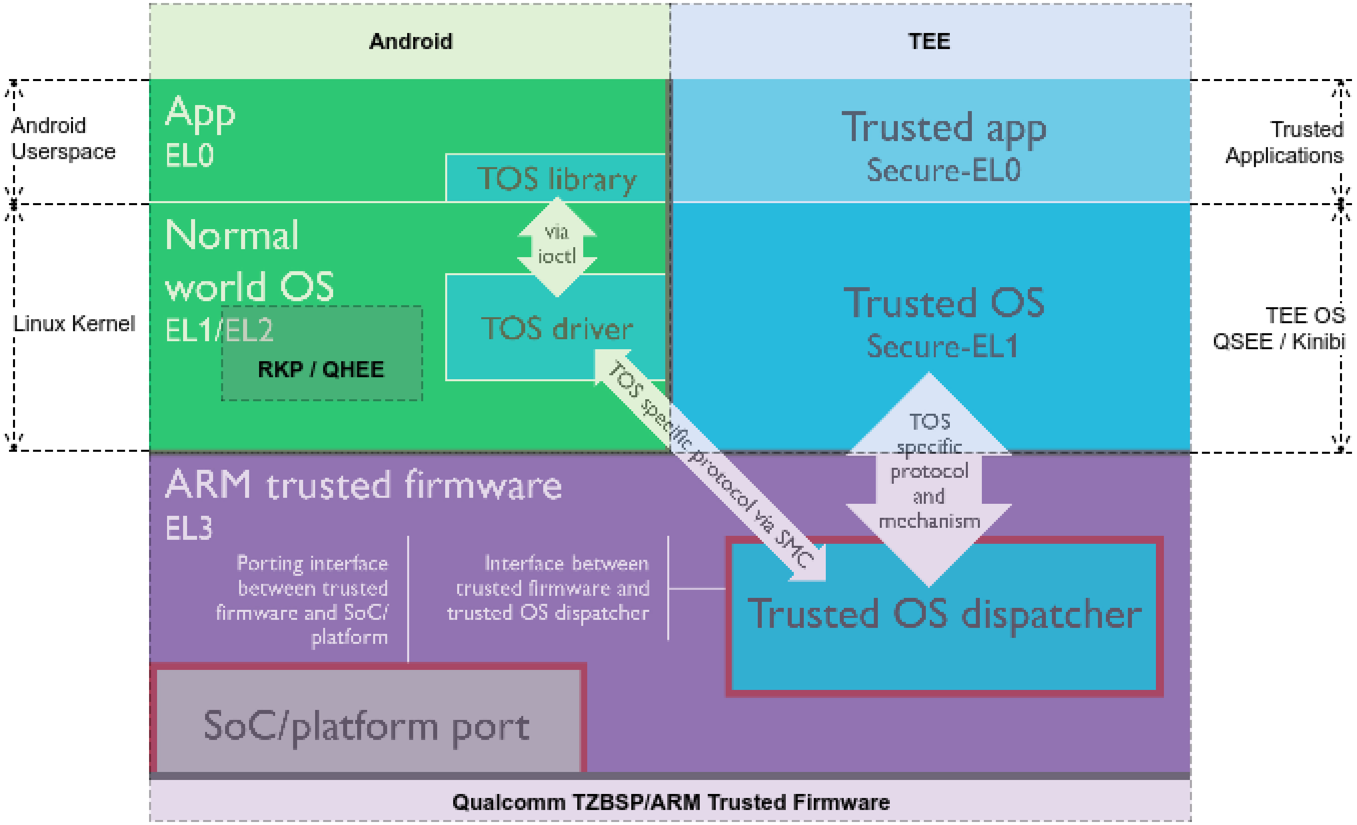
- providing security for the rich environment
- also called an Enclave

More secure than the User-facing OS

- ARM TrustZone TEE: Allows creation of two execution contexts on same resources

TEE will store cryptographic material and hold sensitive applications

- A base concept for mobile payments and secure storage



TEE: Keymaster

Provides access to the keystore

- API based, not full RW access
- Replies to requests from authorized services (shared secret), having a valid (recent) AuthToken

Keymaster 1: Android 6

- Signing API (sign, verify, import keys)

Keymaster 2: Android 7

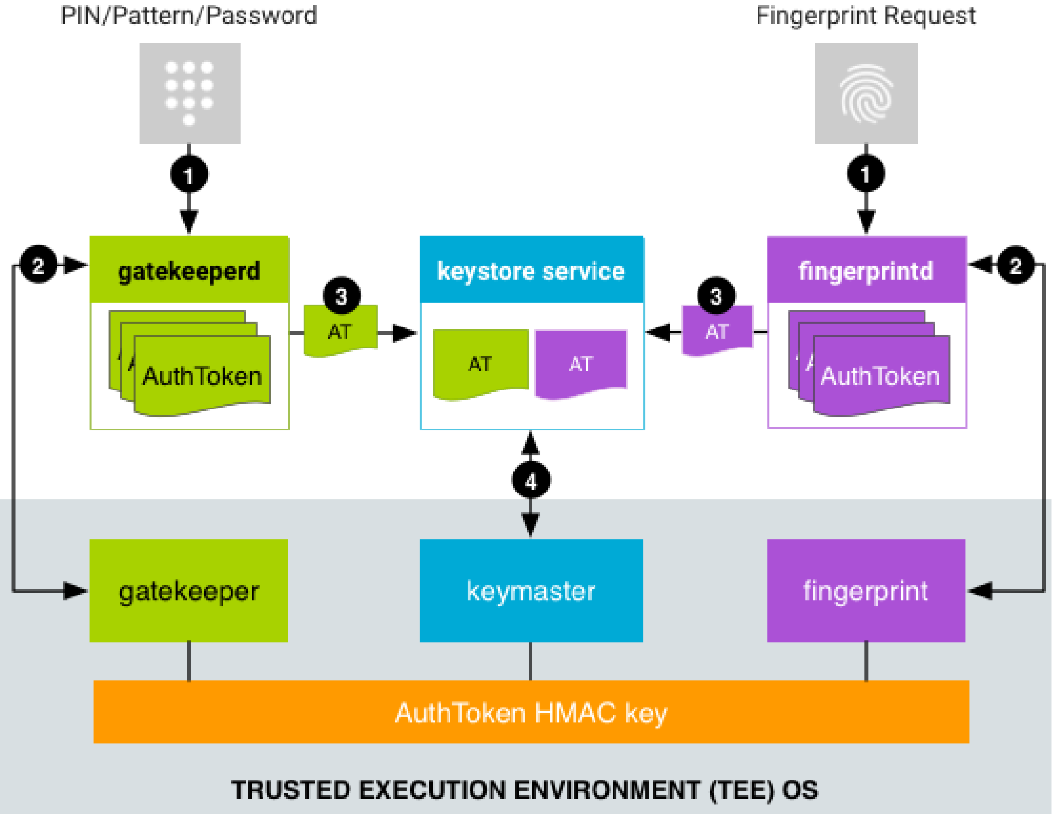
- Support for AES and HMAC
- Key Attestation: Certifies keys (origin, property, usages)
- Version Binding: ties keys to OS and TEE version, preventing downgrades

Keymaster 3: Android 8

- ID Attestation: Key device identifiers are stored as HMAC(HWKEY, IDn)

Keymaster 4: Android 9

- Embedded Secure Elements: allowing embedded "smartcards"



Underlying Platform

Boot is secure with integrity checks by the bootloader

- While this is true, only vendor kernels can be used
- Users may unlock the bootloader allowing to customize the boot process
  - If allowed by the vendor
  - Unlocking will erase all user data

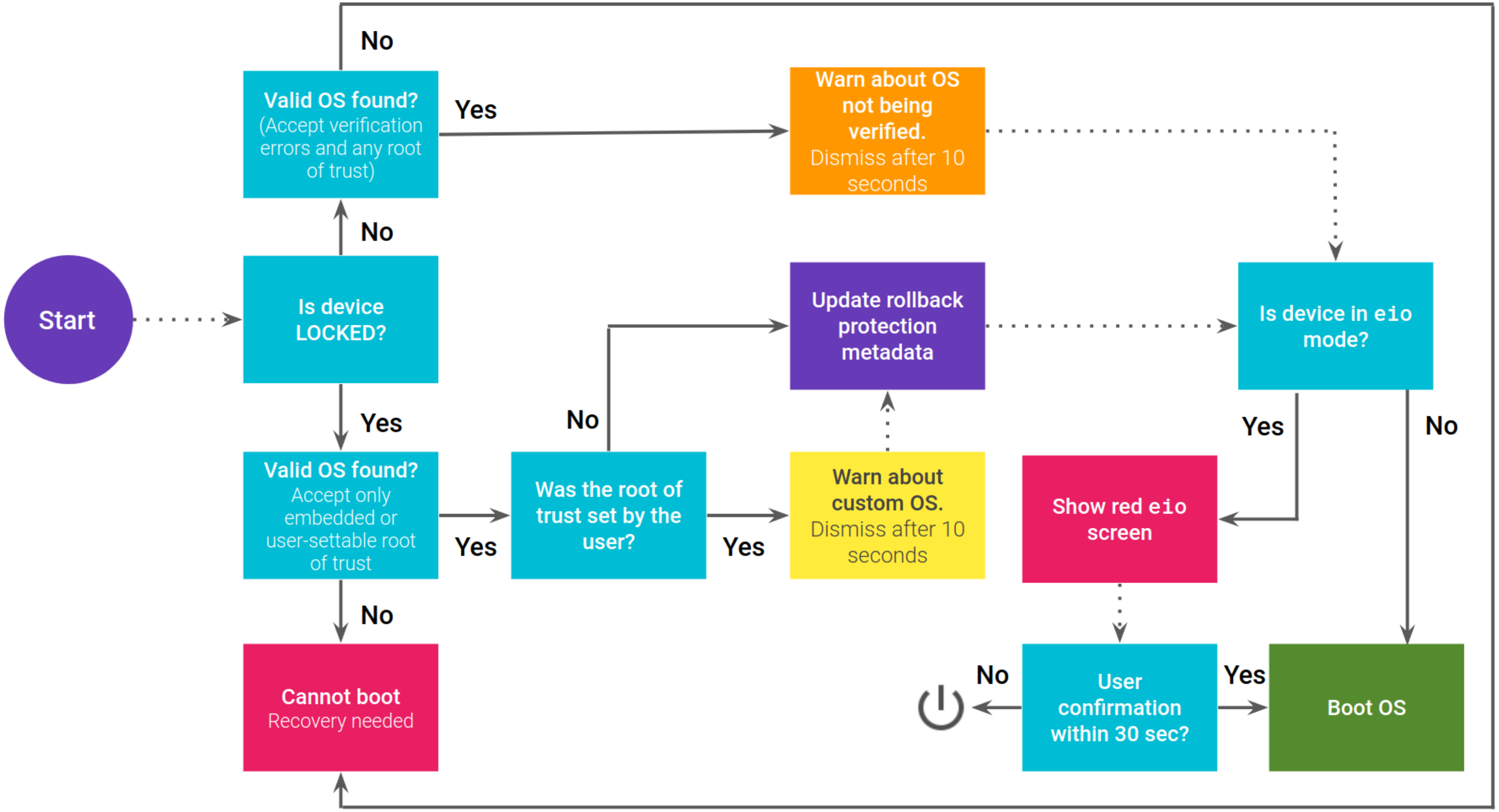
Applications never execute with uid 0 and there is no method of doing it

- Occasionally, attacks to the platform may allow such access
- All interactions are made through the SDK, which run on a Java Virtual Machine

Internal Storage is encrypted

- direct access is not allowed without flashing everything

Underlying Platform



Start
Is device LOCKED?
No
Valid OS found? (Accept verification errors and any root of trust)
No
Yes
Warn about OS not being verified. Dismiss after 10 seconds
Yes
Valid OS found? Accept only embedded or user-settable root of trust
Yes
Was the root of trust set by the user?
No
Update rollback protection metadata
Yes
Warn about custom OS. Dismiss after 10 seconds
Is device in eio mode?
Yes
No
Show red eio screen
No
Cannot boot Recovery needed
No
User confirmation within 30 sec?
Yes
Boot OS

Android Application Permissions

Given the strongly service-based orientation, Access Control is very granular

Applications must declare on compile time which permissions they require

Users may accept the App permissions

- Install Time or at Run Time
- Not granting a permission will effectively block those resources from the App

Typical permissions: Camera, Storage, Contacts, Location, Accessibility, Sensors, SMS, …

```
<manifest ... >
    <uses-permission
android:name="android.permission.SEND_SMS"/>
    ...
</manifest>
```

Android Intents

Intents are a Message Passing mechanism for IPC

- As execution is not persistent and applications are strongly isolated, this provides an effective manner for auditable and controllable IPC

Composed by two main sections

- Action: specifies the action to be triggered. There are several already defined
- Data: specifies the arguments to be passed

Intents can be sent with different scopes

- To all components, to a specific component.
  - Framework will resolve the actual receiver
- Multiple components can receive the same intent
  - We can even have broadcast intents

Mobile security issues

Threat landscape is wide, and attacks are valuable

- A non interaction RCE may award 1-2M€
- A single vulnerability found is immediately applicable to millions of devices

Relevant sources of vulnerabilities

- Underlying software or hardware platform
- **Wrongly coded applications/programming mistakes**
- Abusive applications (malware)
- Users are careless

Attacks can focus on user data, or as a pivot for further actions. Even against support infra.

- Conduct 2FA towards an infrastructure
- Track users and their personal data
- Access bank/financial related data

Platform issues

Vendors follow the design guidelines towards secure systems

- Google enforces minimum security requirements for approved devices

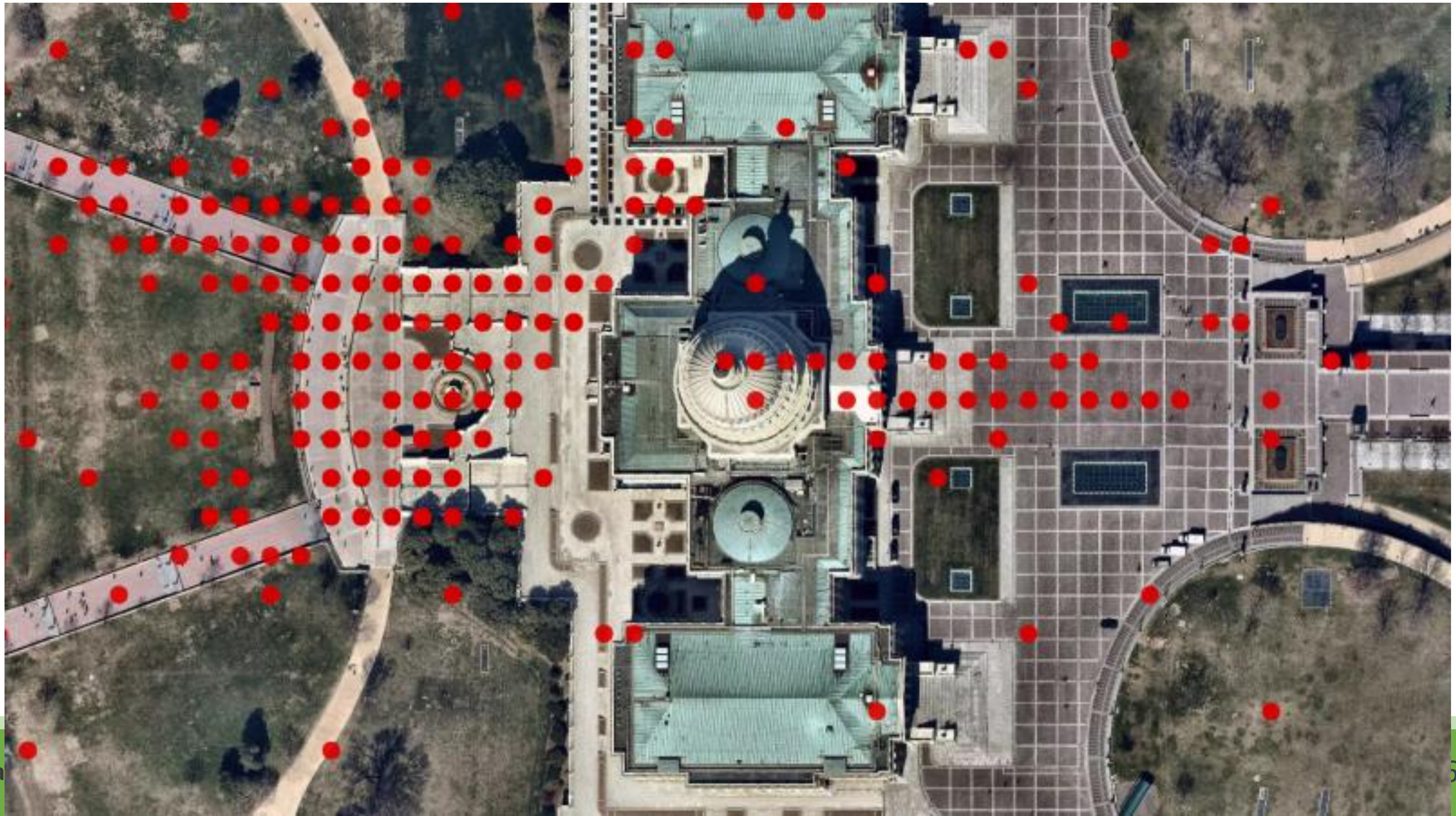
Vendors sometimes also introduce additional issues with their implementations

- Insecure Trustlets in the TEE
  - Cerdeira et al, “SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems” review existing flaws exploiting issues in the TEE
- APDUs for remote management
  - André Pereira et al, “USB connection vulnerabilities on Android smartphones: default and vendors’ customizations” found custom APDUs in Samsung devices disclosing device identification and allowing automated flashing of a malicious app
- Modem implementation
  - QualPwn - Exploiting Qualcomm WLAN and Modem Over The Air
- Vulnerable or abusive pre-installed applications
  - Xiaomi ‘Guard Provider’ downloads antivirus APK through HTTP, allowing remote injection of malicious code

Careless users

Users lack the knowledge to properly assess the impact of providing a permission

- Application may leak data directly, or may use that method to gain additional information



Wrongly coded applications/programming mistakes

Mobile apps are frequently populated with bugs/mistakes as other applications

- Because the code is available to clients, inspection and abuse becomes more frequent
- Java/Kotlin can be decompiled to source code
  - Obfuscation helps but only has limited impact

Mobile app development is popular, with tools providing facilitated access

- Enabling wide use by many developers also increases the amount of security issues
- Being able to implement a mobile app != knowing how to security use the platform
- Mobile apps are used for shop frontends and small trials.
  - There is a respectable amount of sub-quality apps around.

The platform provides some protection mechanisms and scanning for malware

- Yet it doesn't correct bad/naive code

Insecure Bank

A mobile goat application exposing many flaws, for research and training purposes

- Will be used in this class for demonstrating the multiple things that can go wrong

Setup

- Install Android Studio and create a Virtual Device
- Create a Mobile Device emulating a Nexus 5X – API 26
- Install android tools: https://www.xda-developers.com/install-adb-windows-macos-linux/
- Download and install the APK with: `adb install InsecureBankv2.apk`
- You should have a full-blown android device with the application installed
- Download the server code and run it in your PC
- To enable connection between app and server run: adb reverse tcp:8888 tcp:8888
  - This will make the server in the host available in the android using port 8888

Decompiling Mobile Applications

Concepts:

- Disassemble: convert bytecode to Assembly language
- Decompile: convert bytecode to a higher-level representation of the algorithm (Usually a C representation)

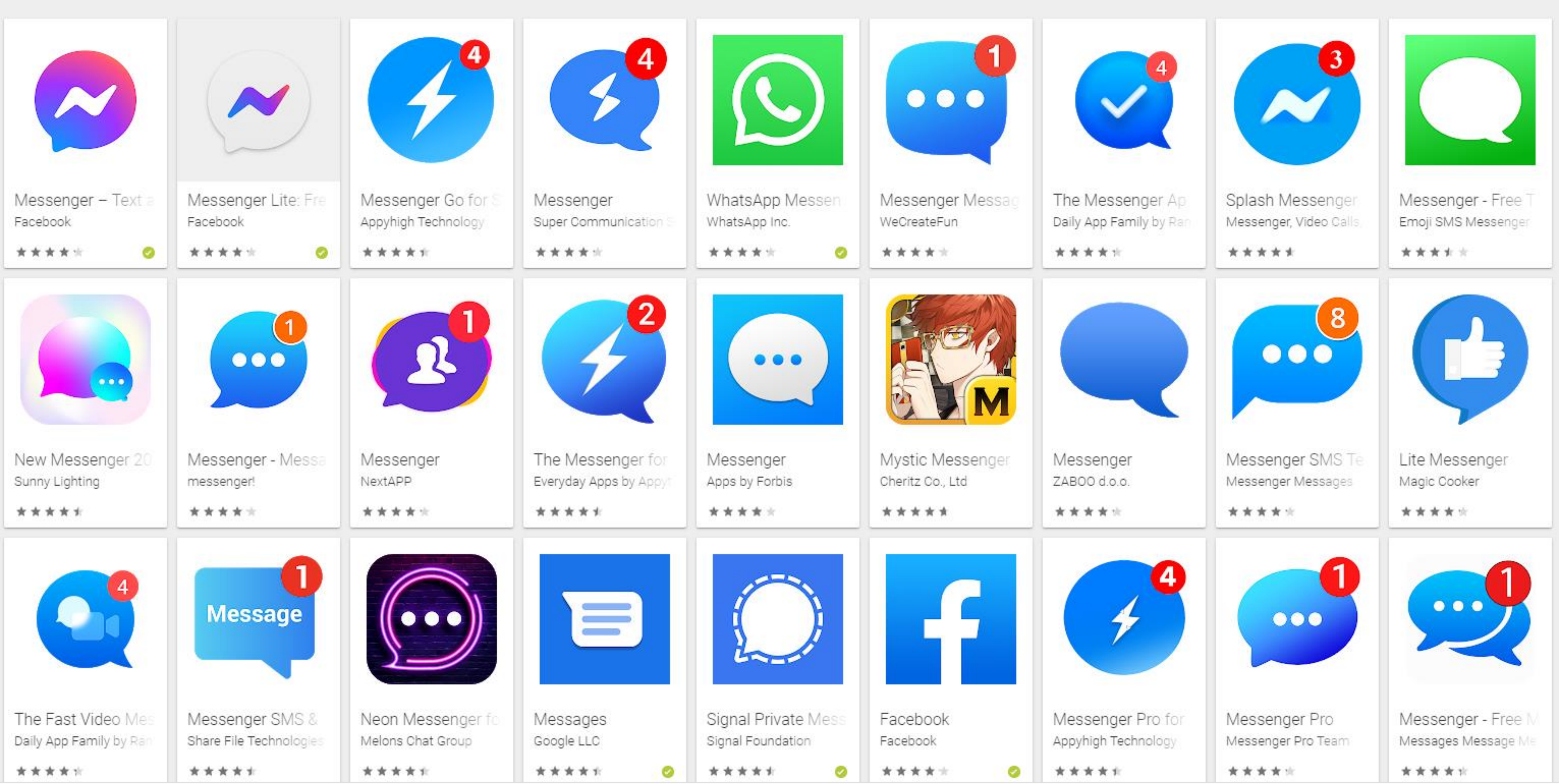
All applications can be analyzed after compilation

- A topic of reverse engineering
- Android applications are particularly susceptible to it as Java bytecode can de decompiled back to Java

Problem: putting too much trust in the "obscurity" provided by bytecode

- An issue for binary applications and even more for android
- Attacker can download, modify, repack and upload an application
- Use of ProGuard or other obfuscation method is still low: https://arxiv.org/pdf/1801.02742.pdf

Impact: manipulation, access to sensitive data, repackage, brand damage



Messenger – Text a
Facebook

Messenger Lite: Fre
Facebook

Messenger Go for S
Appyhigh Technology

Messenger
Super Communication S

WhatsApp Messen
WhatsApp Inc.

Messenger Messag
WeCreateFun

The Messenger Ap
Daily App Family by Ran

Splash Messenger
Messenger, Video Calls

Messenger - Free T
Emoji SMS Messenger

New Messenger 20
Sunny Lighting

Messenger - Messa
messenger!

Messenger
NextAPP

The Messenger for
Everyday Apps by Appy

Messenger
Apps by Forbis

Mystic Messenger
Cheritz Co., Ltd

Messenger
ZABOO d.o.o.

Messenger SMS Te
Messenger Messages

Lite Messenger
Magic Cooker

The Fast Video Mes
Daily App Family by Ran

Messenger SMS &
Share File Technologies

Neon Messenger fo
Melons Chat Group

Messages
Google LLC

Signal Private Mess
Signal Foundation

Facebook
Facebook

Messenger Pro for
Appyhigh Technology

Messenger Pro
Messenger Pro Team

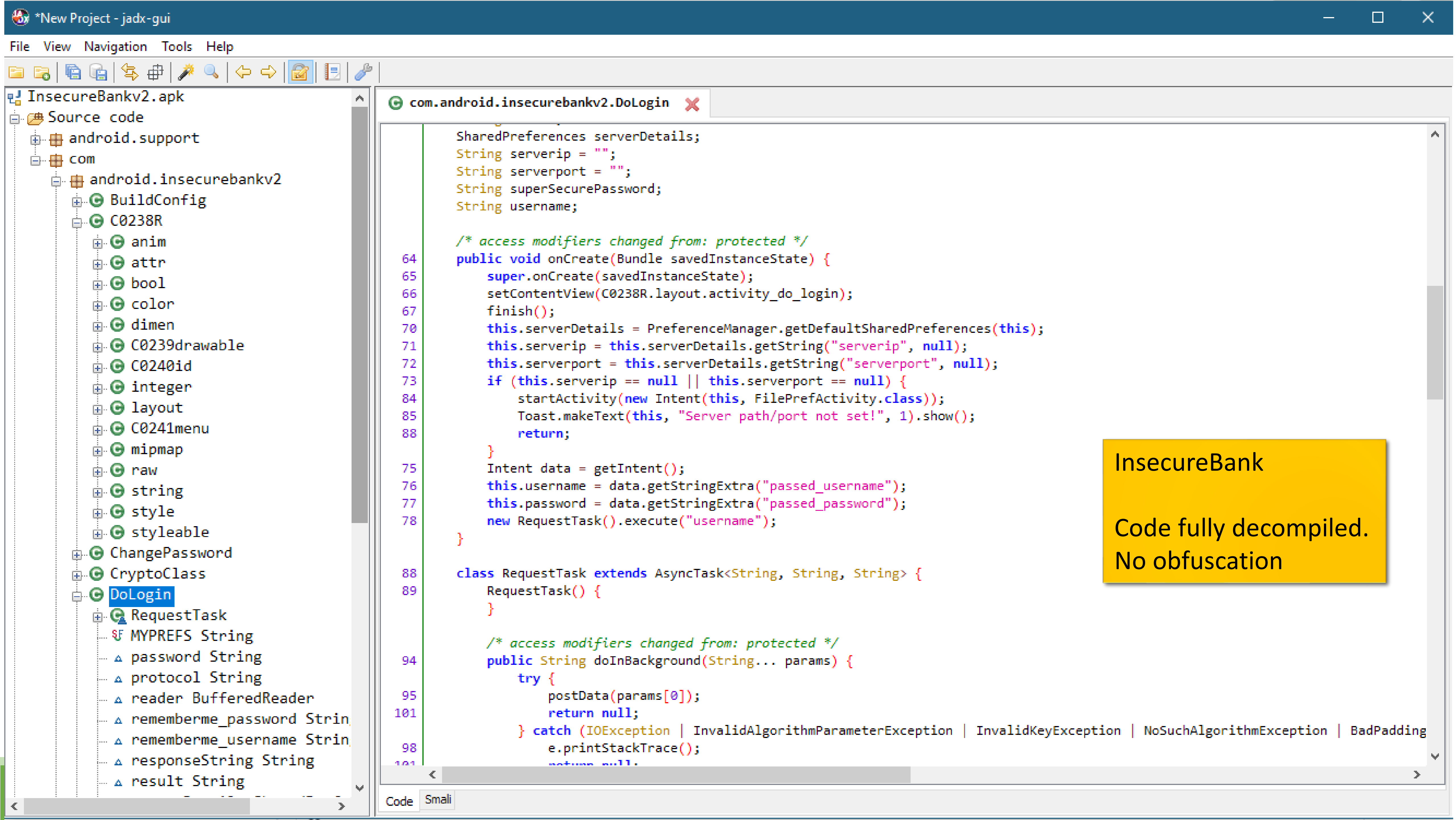
Messenger - Free M
Messages Message Me

Decompiling Mobile Applications

1. **Download InsecureBank.apk**
2. **Download jadx: https://github.com/skylot/jadx**
3. **Open apk with jadx**
4. **Resources and source code should be mostly available**

Remediation: Obfuscators should be used!

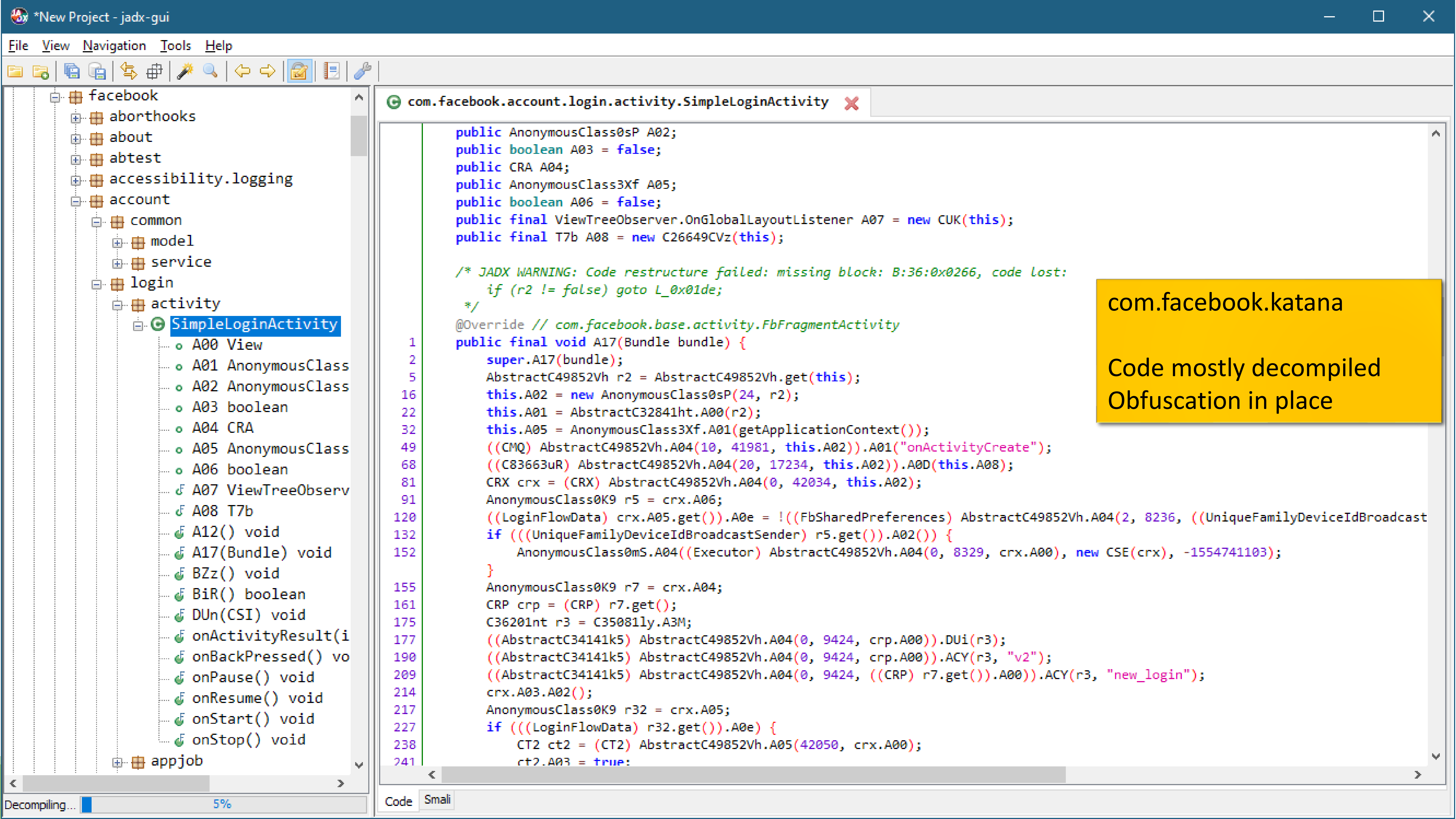
- Remove class names and can rearrange code
- Eliminates dead/unused code
- Can implement anti-decompile mechanisms
- Only increase the effort to decompile an application and do not prevent it



```
    SharedPreferences serverDetails;
    String serverip = "";
    String serverport = "";
    String superSecurePassword;
    String username;

    /* access modifiers changed from: protected */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(C0238R.layout.activity_do_login);
        finish();
        this.serverDetails = PreferenceManager.getDefaultSharedPreferences(this);
        this.serverip = this.serverDetails.getString("serverip", null);
        this.serverport = this.serverDetails.getString("serverport", null);
        if (this.serverip == null || this.serverport == null) {
            startActivity(new Intent(this, FilePrefActivity.class));
            Toast.makeText(this, "Server path/port not set!", 1).show();
            return;
        }
        Intent data = getIntent();
        this.username = data.getStringExtra("passed_username");
        this.password = data.getStringExtra("passed_password");
        new RequestTask().execute("username");
    }

    class RequestTask extends AsyncTask<String, String, String> {
        RequestTask() {
        }

        /* access modifiers changed from: protected */
        public String doInBackground(String... params) {
            try {
                postData(params[0]);
                return null;
            } catch (IOException | InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException | BadPadding
                e.printStackTrace();
                return null;
```

InsecureBank

Code fully decompiled.
No obfuscation



```
    public AnonymousClass0sP A02;
    public boolean A03 = false;
    public CRA A04;
    public AnonymousClass3Xf A05;
    public boolean A06 = false;
    public final ViewTreeObserver.OnGlobalLayoutListener A07 = new CUK(this);
    public final T7b A08 = new C26649CVz(this);

    /* JADX WARNING: Code restructure failed: missing block: B:36:0x0266, code lost:
        if (r2 != false) goto L_0x01de;
     */
    @Override // com.facebook.base.activity.FbFragmentActivity
    public final void A17(Bundle bundle) {
        super.A17(bundle);
        AbstractC49852Vh r2 = AbstractC49852Vh.get(this);
        this.A02 = new AnonymousClass0sP(24, r2);
        this.A01 = AbstractC32841ht.A00(r2);
        this.A05 = AnonymousClass3Xf.A01(getApplicationContext());
        ((CMQ) AbstractC49852Vh.A04(10, 41981, this.A02)).A01("onActivityCreate");
        ((C83663uR) AbstractC49852Vh.A04(20, 17234, this.A02)).A0D(this.A08);
        CRX crx = (CRX) AbstractC49852Vh.A04(0, 42034, this.A02);
        AnonymousClass0K9 r5 = crx.A06;
        ((LoginFlowData) crx.A05.get()).A0e = !((FbSharedPreferences) AbstractC49852Vh.A04(2, 8236, ((UniqueFamilyDeviceIdBroadcast
        if (((UniqueFamilyDeviceIdBroadcastSender) r5.get()).A02()) {
            AnonymousClass0mS.A04((Executor) AbstractC49852Vh.A04(0, 8329, crx.A00), new CSE(crx), -1554741103);
        }
        AnonymousClass0K9 r7 = crx.A04;
        CRP crp = (CRP) r7.get();
        C36201nt r3 = C35081ly.A3M;
        ((AbstractC34141k5) AbstractC49852Vh.A04(0, 9424, crp.A00)).DUi(r3);
        ((AbstractC34141k5) AbstractC49852Vh.A04(0, 9424, crp.A00)).ACY(r3, "v2");
        ((AbstractC34141k5) AbstractC49852Vh.A04(0, 9424, ((CRP) r7.get()).A00)).ACY(r3, "new_login");
        crx.A03.A02();
        AnonymousClass0K9 r32 = crx.A05;
        if (((LoginFlowData) r32.get()).A0e) {
            CT2 ct2 = (CT2) AbstractC49852Vh.A05(42050, crx.A00);
            ct2.A03 = true;
```

com.facebook.katana

Code mostly decompiled
Obfuscation in place

Administrator Interfaces

Mobile applications frequently clients to remote systems

- Similar to what a browser would do
  - Actually, many applications are not more than a web page

However naïve developers may identify an increased security in the use of an APK

- In a web application it is assumed that all code is available to users as HTML/JS
- In a mobile app, everything is enclosed in a APK file

Believing in this and having a wrong sense of security is a serious mistake

Typical issue: inclusion of debug/special access APIs in applications

- Useful for testing purposes
- Left in the application as the developer doesn't expect an attacker to access source code
  - Obfuscation mechanisms presented in most tools actually increase this issue (as they do not work that well)

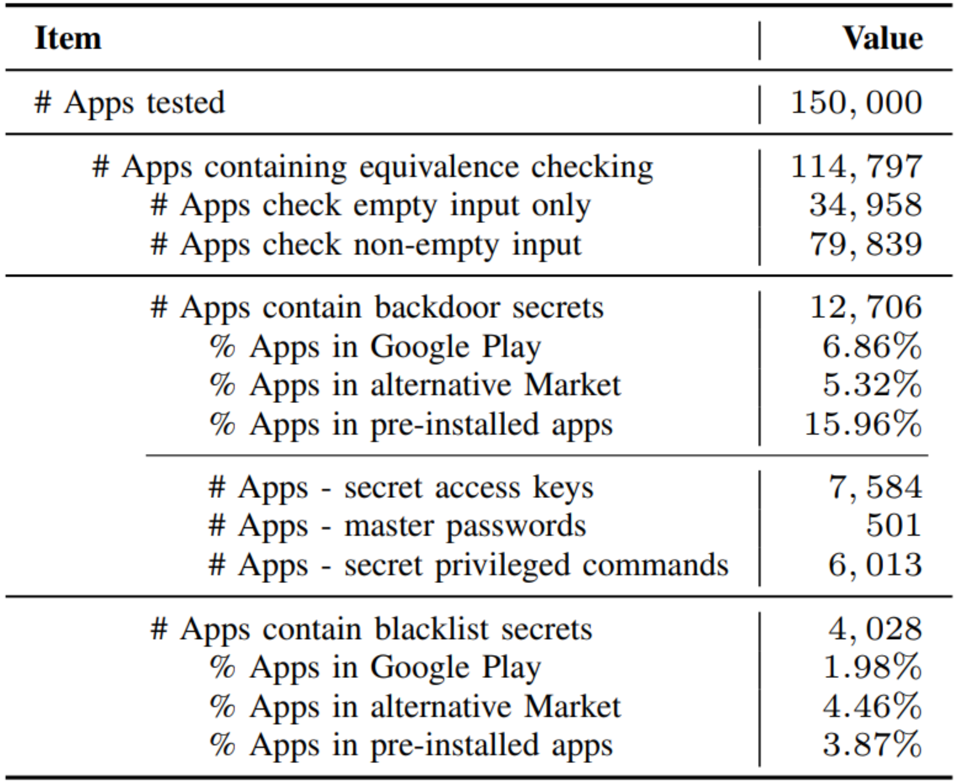
Administrator Interfaces

Issue still affects many applications

- Interestingly, mostly pre-installed apps!
  - Which users cannot uninstall and have large install

Access to such interfaces may provide access beyond expectations

- May circumvent further access control



Item	Value
# Apps tested	150,000
# Apps containing equivalence checking	114,797
# Apps check empty input only	34,958
# Apps check non-empty input	79,839
# Apps contain backdoor secrets	12,706
% Apps in Google Play	6.86%
% Apps in alternative Market	5.32%
% Apps in pre-installed apps	15.96%
# Apps - secret access keys	7,584
# Apps - master passwords	501
# Apps - secret privileged commands	6,013
# Apps contain blacklist secrets	4,028
% Apps in Google Play	1.98%
% Apps in alternative Market	4.46%
% Apps in pre-installed apps	3.87%

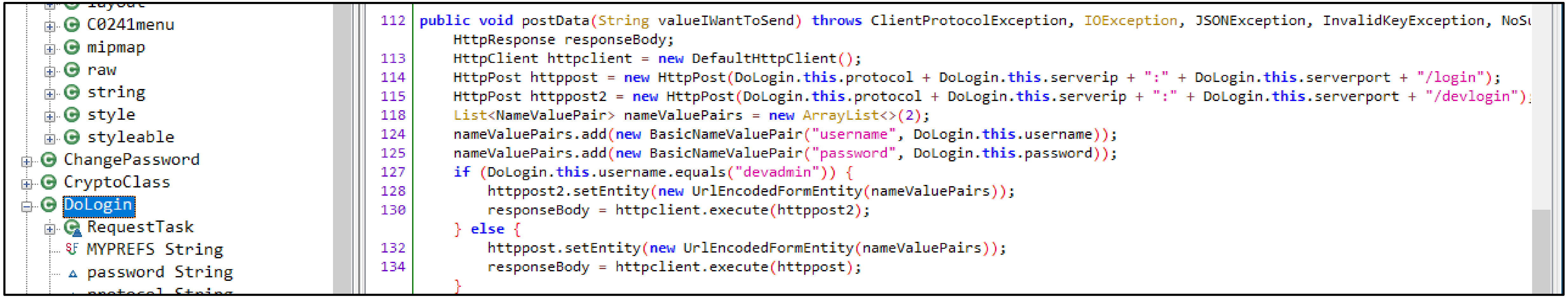
Qingchuan Zhao, Chaoshun Zuo, Brendan Dolan-Gavitt, Giancarlo Pellegrino , Zhiqiang Lin
“Automatic Uncovering of Hidden Behaviors From Input Validation in Mobile Apps”

Administrator Interfaces

Exercise: can you find a hardcoded login in the bank app?

- What was the purpose of adding said interfaces?
- What impact can be expected?
- Are they required?

Administrator Interfaces



```
public void postData(String valueIWantToSend) throws ClientProtocolException, IOException, JSONException, InvalidKeyException, NoSu
    HttpResponse responseBody;
    HttpClient httpclient = new DefaultHttpClient();
    HttpPost httppost = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":" + DoLogin.this.serverport + "/login");
    HttpPost httppost2 = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":" + DoLogin.this.serverport + "/devlogin");
    List<NameValuePair> nameValuePairs = new ArrayList<>(2);
    nameValuePairs.add(new BasicNameValuePair("username", DoLogin.this.username));
    nameValuePairs.add(new BasicNameValuePair("password", DoLogin.this.password));
    if (DoLogin.this.username.equals("devadmin")) {
        httppost2.setEntity(new UrlEncodedFormEntity(nameValuePairs));
        responseBody = httpclient.execute(httppost2);
    } else {
        httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
        responseBody = httpclient.execute(httppost);
    }
```

Alternative login uses a different login process if username="devadmin"

- /devlogin instead of /login

Impact: User devadmin provides access no matter what the password is

- Probably a left over from the development process

Hardcoded secrets

May be related to the existence of administrator interfaces

- Credentials to access the hidden API

May be related to other functionality, such as poorly implemented secure storage

- Using shared preferences or files to store sensitive material

Vuln. consists of not using hardware backed storage to store keys

- If they are in code, they can be obtained by decompilation
  - they should be considered as public as an attacker may access them any time
- More common on older implementations targeting devices without an advanced TEE

Solution: good code practices and secret detection tools

- Automated tools (GitGuardian, truffleHog) may analyze repositories and trigger alarms automatically

Exercise: Search the Insecure Bank application for hardcoded secrets. Can you find them?

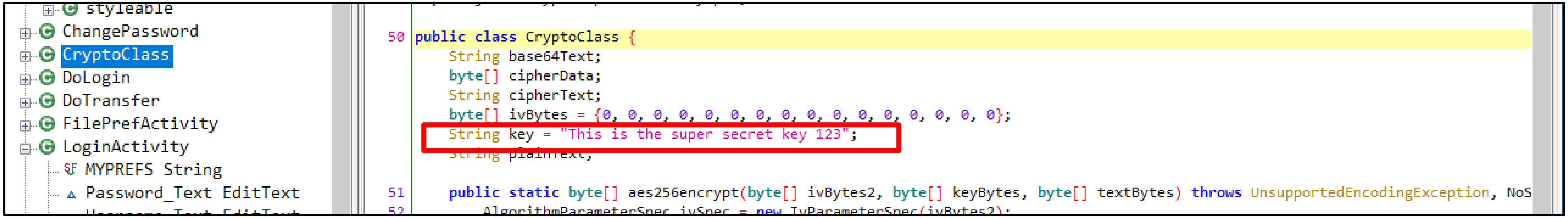
- What is the impact of said hardcoded secrets?

Hardcoded secrets

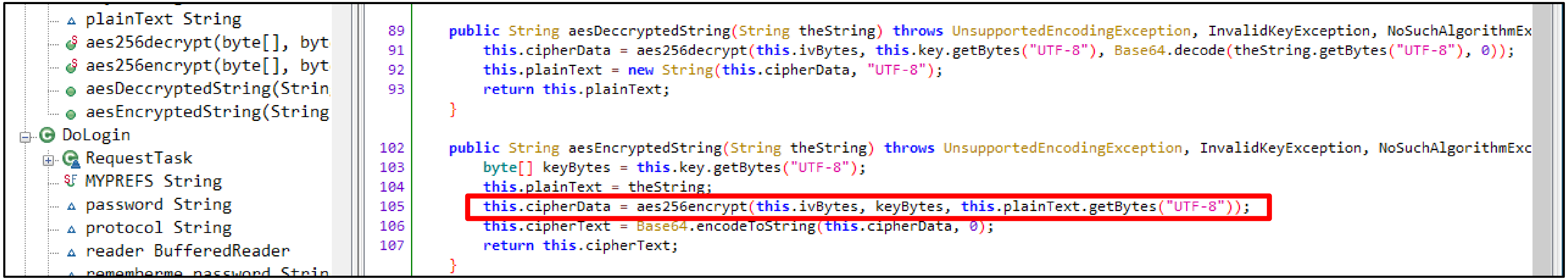
Exercise: Search the Insecure Bank application for hardcoded secrets.

- What is the impact of said hardcoded secrets?
- Why are they there?
- How could they be avoided?

Hardcoded secrets



```
public class CryptoClass {
    String base64Text;
    byte[] cipherData;
    String cipherText;
    byte[] ivBytes = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    String key = "This is the super secret key 123";
    String plainText;

    public static byte[] aes256encrypt(byte[] ivBytes2, byte[] keyBytes, byte[] textBytes) throws UnsupportedEncodingException, NoS
        AlgorithmParameterSpec ivSpec = new IvParameterSpec(ivBytes2);
```


plainText String
aes256decrypt(byte[], byt
aes256encrypt(byte[], byt
aesDeccryptedString(Strin
aesEncryptedString(String
DoLogin
RequestTask
MYPREFS String
password String
protocol String
reader BufferedReader
rememberme_password Strin

```
    public String aesDeccryptedString(String theString) throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmEx
        this.cipherData = aes256decrypt(this.ivBytes, this.key.getBytes("UTF-8"), Base64.decode(theString.getBytes("UTF-8"), 0));
        this.plainText = new String(this.cipherData, "UTF-8");
        return this.plainText;
    }

    public String aesEncryptedString(String theString) throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmExc
        byte[] keyBytes = this.key.getBytes("UTF-8");
        this.plainText = theString;
        this.cipherData = aes256encrypt(this.ivBytes, keyBytes, this.plainText.getBytes("UTF-8"));
        this.cipherText = Base64.encodeToString(this.cipherData, 0);
        return this.cipherText;
    }
```

A hardcoded constant is available on the code, used to encrypt/decrypt strings

Impact: while vendor will advertise that passwords are stored with AES-256, they are not securely stored

Visibility Issues

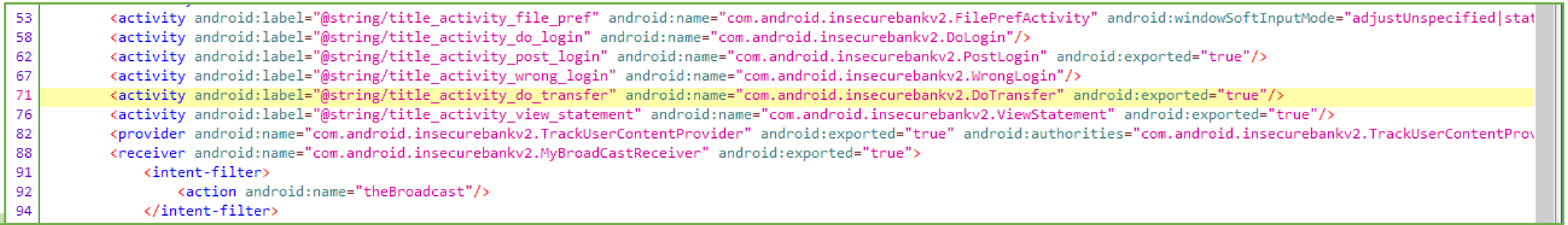
Activities are usually internal to an application

- Called as the standard interaction workflow

Activities can be made available to be called directly

- Provides additional entry points to the application
- Should never be done for internal activities without further access control
  - Developers may set activities as exported for debugging purposes
  - Failure to remove such property may allow circumvention of the proper app operation

Activity visibility is set in the AndroidManifest.xml at compile time



```
<activity android:label="@string/title_activity_file_pref" android:name="com.android.insecurebankv2.FilePrefActivity" android:windowSoftInputMode="adjustUnspecified|stat
<activity android:label="@string/title_activity_do_login" android:name="com.android.insecurebankv2.DoLogin"/>
<activity android:label="@string/title_activity_post_login" android:name="com.android.insecurebankv2.PostLogin" android:exported="true"/>
<activity android:label="@string/title_activity_wrong_login" android:name="com.android.insecurebankv2.WrongLogin"/>
<activity android:label="@string/title_activity_do_transfer" android:name="com.android.insecurebankv2.DoTransfer" android:exported="true"/>
<activity android:label="@string/title_activity_view_statement" android:name="com.android.insecurebankv2.ViewStatement" android:exported="true"/>
<provider android:name="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true" android:authorities="com.android.insecurebankv2.TrackUserContentProv
<receiver android:name="com.android.insecurebankv2.MyBroadCastReceiver" android:exported="true">
    <intent-filter>
        <action android:name="theBroadcast"/>
    </intent-filter>
```

Visibility Issues

Exercise: Explore exported activities in the Insecure Bank app

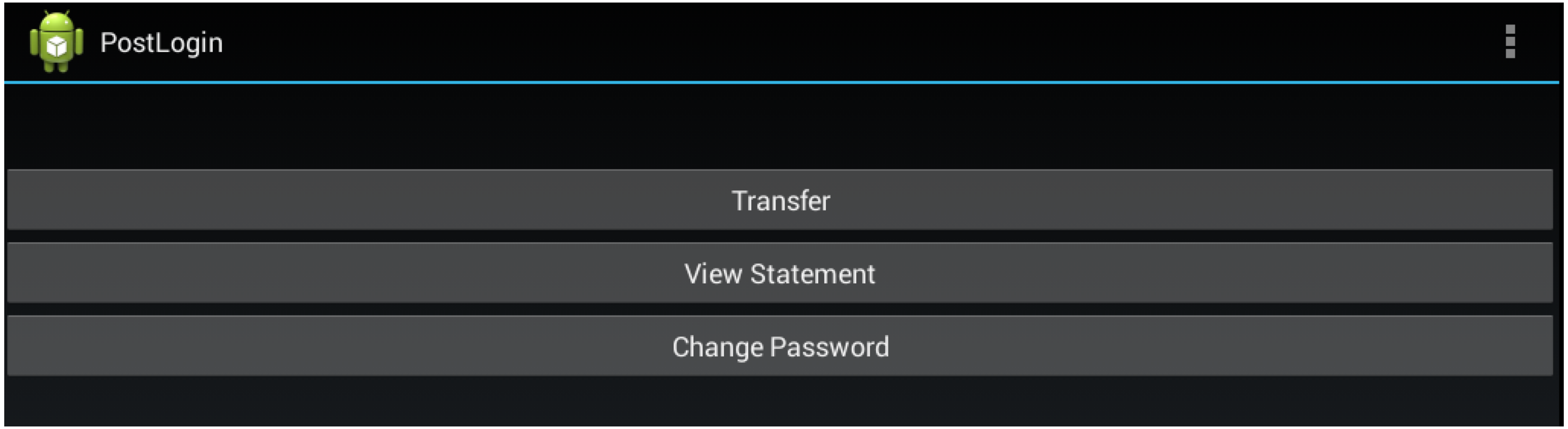
- Which activities are available?
- Do they provide critical functionality without control?
- Test the activities available: “adb shell am start -n com.android.insecurebankv2/com.android.insecurebankv2.ACTIVITY_NAME”

- You may also use drozer
  - Agent: https://github.com/mwrlabs/drozer/releases/download/2.3.4/drozer-agent-2.3.4.apk
  - Server: docker run -it kengannonmwr/drozer_docker
  - Then:
    - Start drozer agent on mobile environment
    - adb forward tcp:31415 tcp:31415
    - docker run -it kengannonmwr/drozer_docker
    - drozer console connect –server ANDROID_IP_ADDRESS
      - run app.package.list
      - run app.package.info -a com.android.insecurebankv2
      - run app.package.attacksurface com.android.insecurebankv2
      - run app.activity.start --component com.android.insecurebankv2 com.android.insecurebankv2.ACTIVITY_NAME

Visibility Issues

Exercise: Explore exported activities in the Insecure Bank app

- Which activities are available?
- Do they provide critical functionality without control?
- Test the activities available:
  - adb shell am start -n activity_name
  - run app.activity.start activity_name



Content Provider Exposure

Content providers enable components to query data

- They abstract internal data management process and expose data by request
  - Methods: query(), insert(), update(), delete()
- Similar to activities, if they are exported, data is available to other applications

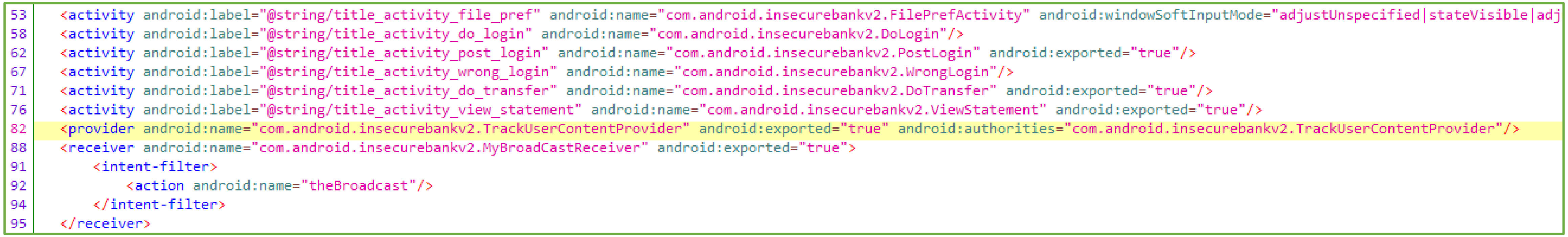
Further access control mechanisms can be used:

- android:permission – provides specific access with good granularity (Read vs Write)
- android:path="/subpath": access can be restricted to a specific set of data
- Temporary permissions: Applications may grant access to others in runtime
  - Ex: upon receiving a broadcast intent stating that a friendly application is installed and was started

```
<provider …>
…
<path-permission android:pathPrefix=“/subpath1” android:readPermission="com.app.SUBPATH1_READ_PERMISSION“ android:writePermission="com.app.SUBPATH1_WRITE_PERMISSION" />
<path-permission android:pathPrefix=“/subpath2” android:readPermission="com.app.SUBPATH2_READ_PERMISSION" android:writePermission="com.app.SUBPATH2_WRITE_PERMISSION" />

<grant-uri-permission android:path=“/subpath2”
</provider>
```

Content Provider Exposure

Exercise: Interbank has one content provider



```
<activity android:label="@string/title_activity_file_pref" android:name="com.android.insecurebankv2.FilePrefActivity" android:windowSoftInputMode="adjustUnspecified|stateVisible|adj
<activity android:label="@string/title_activity_do_login" android:name="com.android.insecurebankv2.DoLogin"/>
<activity android:label="@string/title_activity_post_login" android:name="com.android.insecurebankv2.PostLogin" android:exported="true"/>
<activity android:label="@string/title_activity_wrong_login" android:name="com.android.insecurebankv2.WrongLogin"/>
<activity android:label="@string/title_activity_do_transfer" android:name="com.android.insecurebankv2.DoTransfer" android:exported="true"/>
<activity android:label="@string/title_activity_view_statement" android:name="com.android.insecurebankv2.ViewStatement" android:exported="true"/>
<provider android:name="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true" android:authorities="com.android.insecurebankv2.TrackUserContentProvider"/>
<receiver android:name="com.android.insecurebankv2.MyBroadCastReceiver" android:exported="true">
    <intent-filter>
        <action android:name="theBroadcast"/>
    </intent-filter>
</receiver>
```

Check the implementation what action is triggered, and which data is provided

- You can query it with :
  - `adb shell content query --uri content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers`
  - `run app.provider.query content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers`

Intent based attacks

Intents are the basic mechanism of IPC within applications

- Consist of messages sent between components
- Intents may be broadcasted or explicit
- Intents may be subscribed to by components, even if from other applications
- Providers and receivers are declared in the AndroidManifest.xml
  - Attackers can rapidly check which code may be vulnerable

Correct use of intents allows applications to trigger actions in response to events

- Examples: Show a popup, show an activity, trigger a synchronization process…

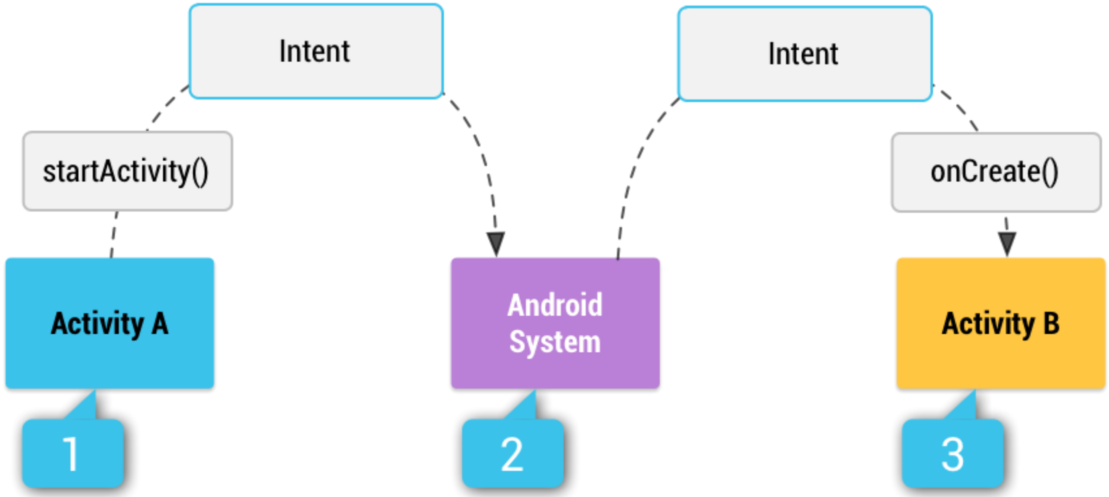
Bad use of intents allow attacker to:

- Intent Sniffing: Gain additional access to confidential data by sniffing intents exchanged by applications
- Intent Spoofing: Trigger specific processes in applications
  - Potentially fuzz arguments or inject malicious payloads
  - Potentially bypassing internal processes and controls

Intent based attacks

Implicit Intents: Extensively used to trigger events based on device state change

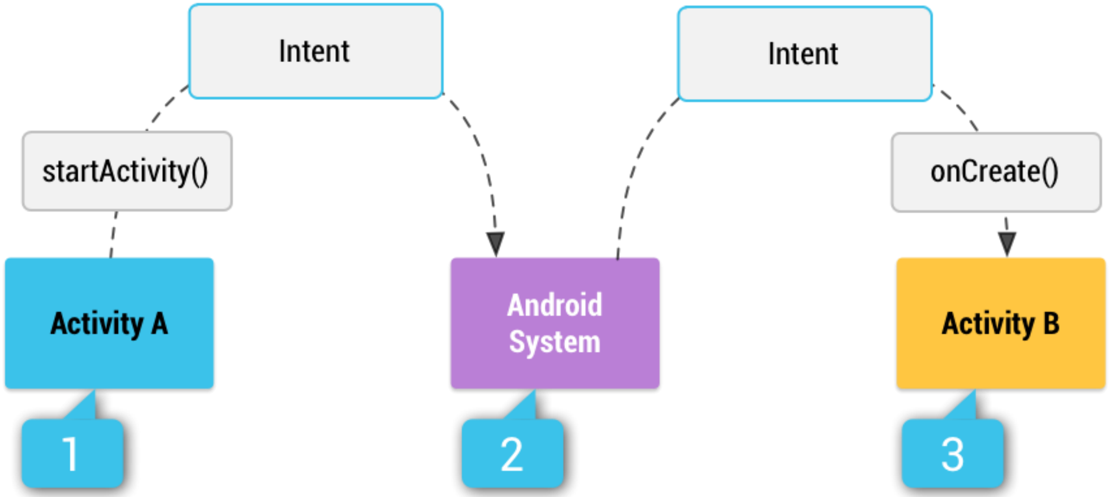
- Intents are sent to all applications with a matching receiver (Broadcasted)
- Specify an action: NETWORK_STATE_CHANGED_ACTION, ACTION_AIRPLANE_MODE_CHANGED…
- They do not specify a destination component
- They should not have sensitive data
- However,… they are the easiest to implement as developers can struggle with when a specific component is specified

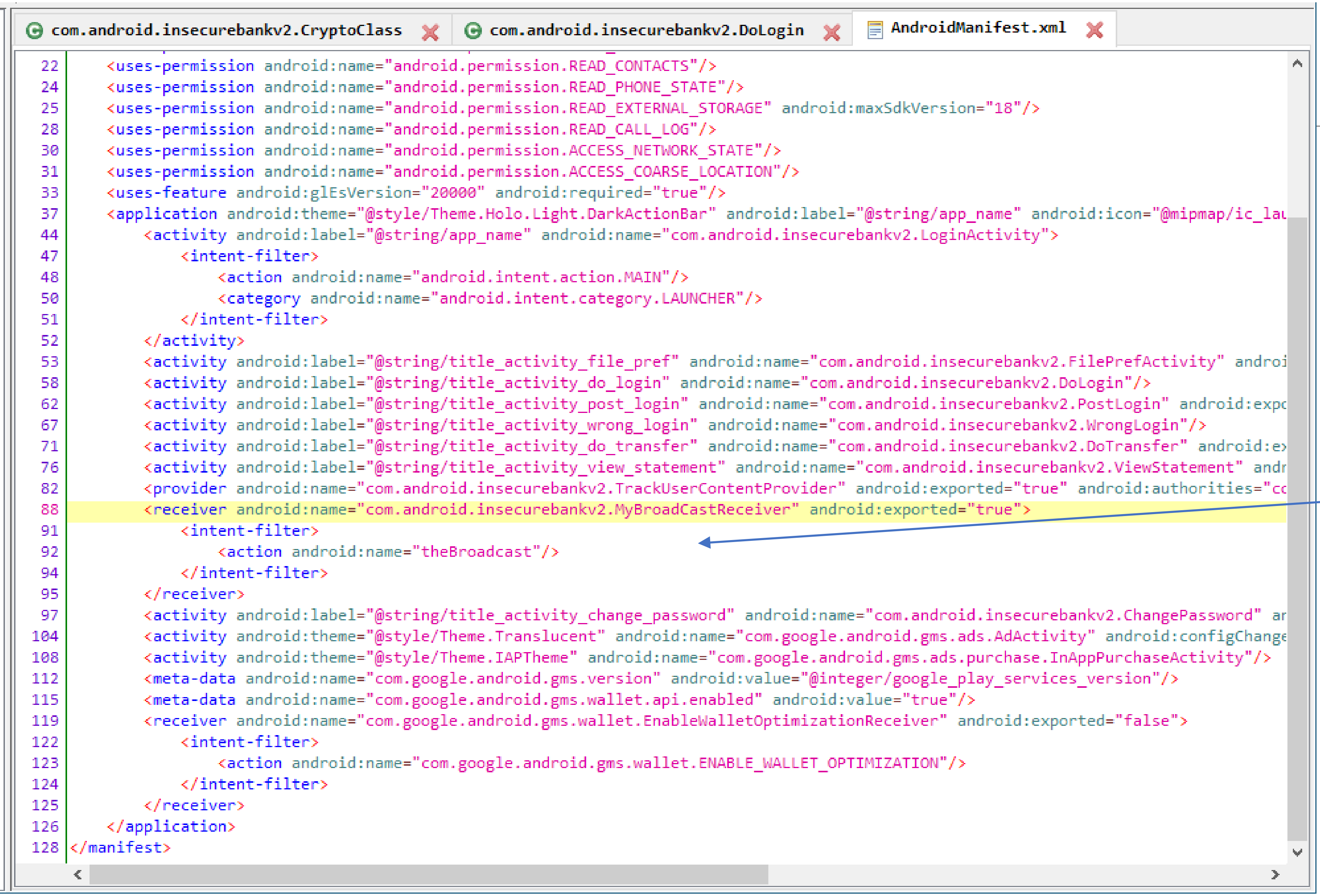


Intent based attacks

Explicit Intents: Used for IPC directly between known components

- Intents are sent to destinations with a matching component
- They can have sensitive data
- However… they are more complex to implement as they require knowledge of the destination component



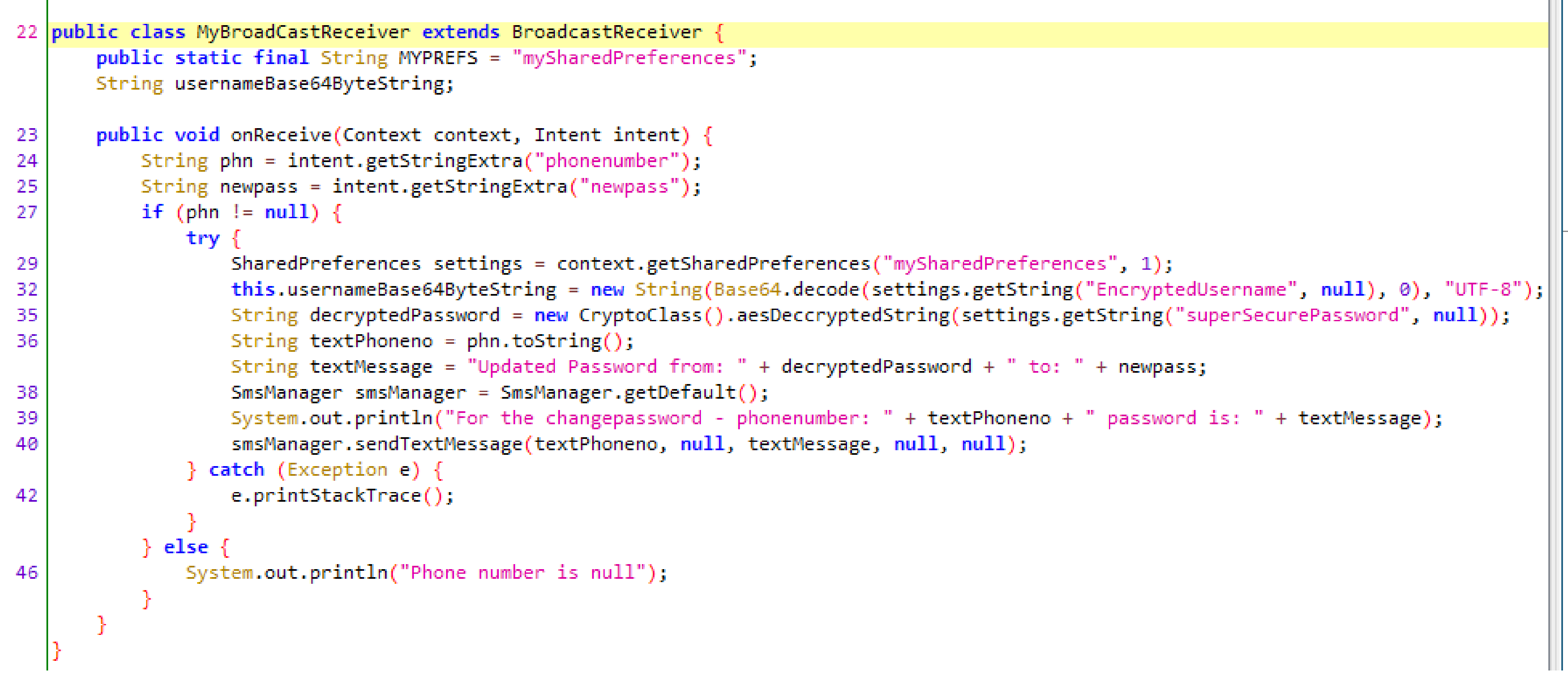


com.android.insecurebankv2.CryptoClass
com.android.insecurebankv2.DoLogin
AndroidManifest.xml

```
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" android:maxSdkVersion="18"/>
    <uses-permission android:name="android.permission.READ_CALL_LOG"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-feature android:glEsVersion="20000" android:required="true"/>
    <application android:theme="@style/Theme.Holo.Light.DarkActionBar" android:label="@string/app_name" android:icon="@mipmap/ic_lau
        <activity android:label="@string/app_name" android:name="com.android.insecurebankv2.LoginActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:label="@string/title_activity_file_pref" android:name="com.android.insecurebankv2.FilePrefActivity" androi
        <activity android:label="@string/title_activity_do_login" android:name="com.android.insecurebankv2.DoLogin"/>
        <activity android:label="@string/title_activity_post_login" android:name="com.android.insecurebankv2.PostLogin" android:expo
        <activity android:label="@string/title_activity_wrong_login" android:name="com.android.insecurebankv2.WrongLogin"/>
        <activity android:label="@string/title_activity_do_transfer" android:name="com.android.insecurebankv2.DoTransfer" android:ex
        <activity android:label="@string/title_activity_view_statement" android:name="com.android.insecurebankv2.ViewStatement" andr
        <provider android:name="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true" android:authorities="co
        <receiver android:name="com.android.insecurebankv2.MyBroadCastReceiver" android:exported="true">
            <intent-filter>
                <action android:name="theBroadcast"/>
            </intent-filter>
        </receiver>
        <activity android:label="@string/title_activity_change_password" android:name="com.android.insecurebankv2.ChangePassword" ar
        <activity android:theme="@style/Theme.Translucent" android:name="com.google.android.gms.ads.AdActivity" android:configChange
        <activity android:theme="@style/Theme.IAPTheme" android:name="com.google.android.gms.ads.purchase.InAppPurchaseActivity"/>
        <meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version"/>
        <meta-data android:name="com.google.android.gms.wallet.api.enabled" android:value="true"/>
        <receiver android:name="com.google.android.gms.wallet.EnableWalletOptimizationReceiver" android:exported="false">
            <intent-filter>
                <action android:name="com.google.android.gms.wallet.ENABLE_WALLET_OPTIMIZATION"/>
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

A receiver is declared and exported

- If it was not exported, declaring an intent-filter will export it (danger)
- Any application may send an intent to this receiver



```java
public class MyBroadCastReceiver extends BroadcastReceiver {
    public static final String MYPREFS = "mySharedPreferences";
    String usernameBase64ByteString;

    public void onReceive(Context context, Intent intent) {
        String phn = intent.getStringExtra("phonenumber");
        String newpass = intent.getStringExtra("newpass");
        if (phn != null) {
            try {
                SharedPreferences settings = context.getSharedPreferences("mySharedPreferences", 1);
                this.usernameBase64ByteString = new String(Base64.decode(settings.getString("EncryptedUsername", null), 0), "UTF-8");
                String decryptedPassword = new CryptoClass().aesDeccryptedString(settings.getString("superSecurePassword", null));
                String textPhoneno = phn.toString();
                String textMessage = "Updated Password from: " + decryptedPassword + " to: " + newpass;
                SmsManager smsManager = SmsManager.getDefault();
                System.out.println("For the changepassword - phonenumber: " + textPhoneno + " password is: " + textMessage);
                smsManager.sendTextMessage(textPhoneno, null, textMessage, null, null);
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            System.out.println("Phone number is null");
        }
    }
}
```

onReceive() lacks validation, assumes two Strings in the intent and triggers an action

As an Intent is an IPC open to external entities, its content should not be trusted

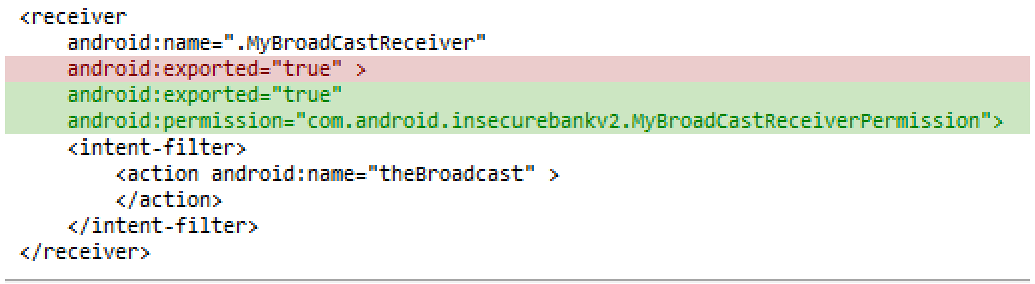
- Fields may be missing
- Fields may have malicious payloads and even trigger further vulnerabilities
  - Raimondas Sasnauskas, “Intent Fuzzer: Crafting Intents of Death”, Proceedings of the 2014 Joint International Workshop on Dynamic Analysis (WODA) and Software and System Performance Testing, Debugging, and Analytics (PERTEA)July 2014
- May also be relevant to check the intent source
- Additional authentication mechanisms can be added to intents: signatures and permissions

Intent based attacks

Exercise: Explore how intent based attacks can be exploited in this app

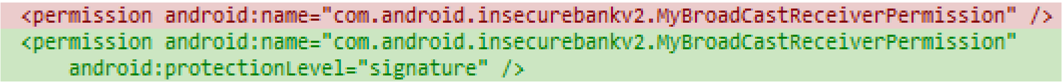
- Drozer:
  - Battery: run app.broadcast.sniff --action android.intent.action.BATTERY_CHANGED
  - Bank app: run app.broadcast.sniff --action "theBroadcast"
  - run app.broadcast.send --action theBroadcast --extra string ARG VAL

Fix 1 – Permission



```
<receiver
    android:name=".MyBroadCastReceiver"
    android:exported="true" >
    android:exported="true"
    android:permission="com.android.insecurebankv2.MyBroadCastReceiverPermission">
    <intent-filter>
        <action android:name="theBroadcast" >
        </action>
    </intent-filter>
</receiver>
```

Fix 2 – Signature



```
 <permission android:name="com.android.insecurebankv2.MyBroadCastReceiverPermission" />
 <permission android:name="com.android.insecurebankv2.MyBroadCastReceiverPermission"
     android:protectionLevel="signature" />
```

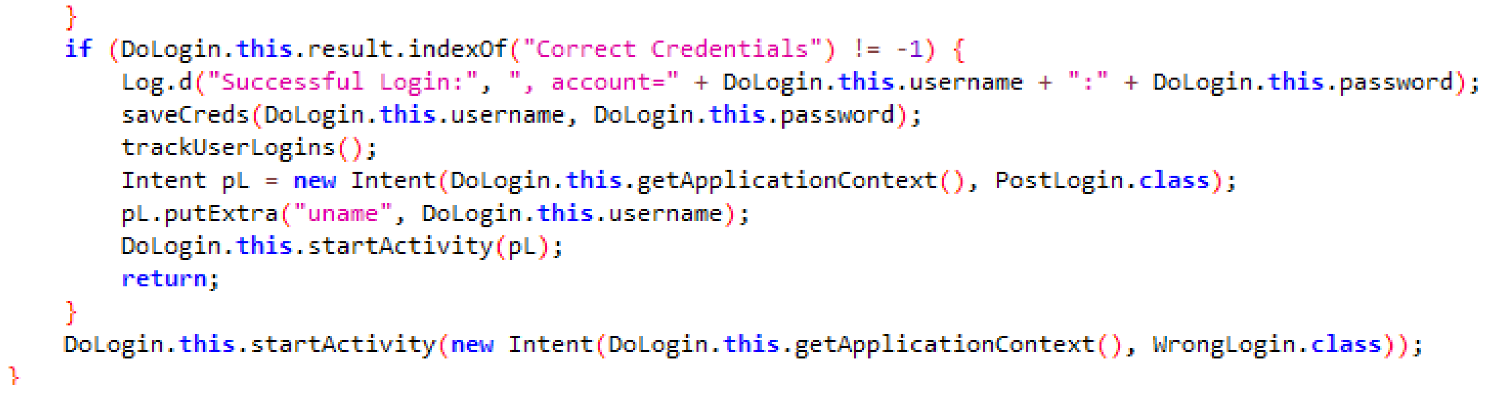
Insecure Logging mechanism

Android has a centralized log to where applications may write information

- Useful for debugging and tracking errors, mostly useless for common users
- Left over debugging lines in code may expose too much information
- Accessible to applications in rooted devices and using `adb logcat`
  - On rooted devices: pm grant <pkg> android.permission.READ_LOGS

Impact:

- Sensitive information is exposed to applications or external attackers



```
    }
    if (DoLogin.this.result.indexOf("Correct Credentials") != -1) {
        Log.d("Successful Login:", ", account=" + DoLogin.this.username + ":" + DoLogin.this.password);
        saveCreds(DoLogin.this.username, DoLogin.this.password);
        trackUserLogins();
        Intent pL = new Intent(DoLogin.this.getApplicationContext(), PostLogin.class);
        pL.putExtra("uname", DoLogin.this.username);
        DoLogin.this.startActivity(pL);
        return;
    }
    DoLogin.this.startActivity(new Intent(DoLogin.this.getApplicationContext(), WrongLogin.class));
}
```

Insecure Logging mechanism

Exercise: use adb logcat and search for sensible strings

- Interact with the applications to observe logs
- What is the impact?

Exercise

Can you replicate these methods to other applications publicly available?

UA Mobile?

CantinUA?

CM Aveiro?

Others?