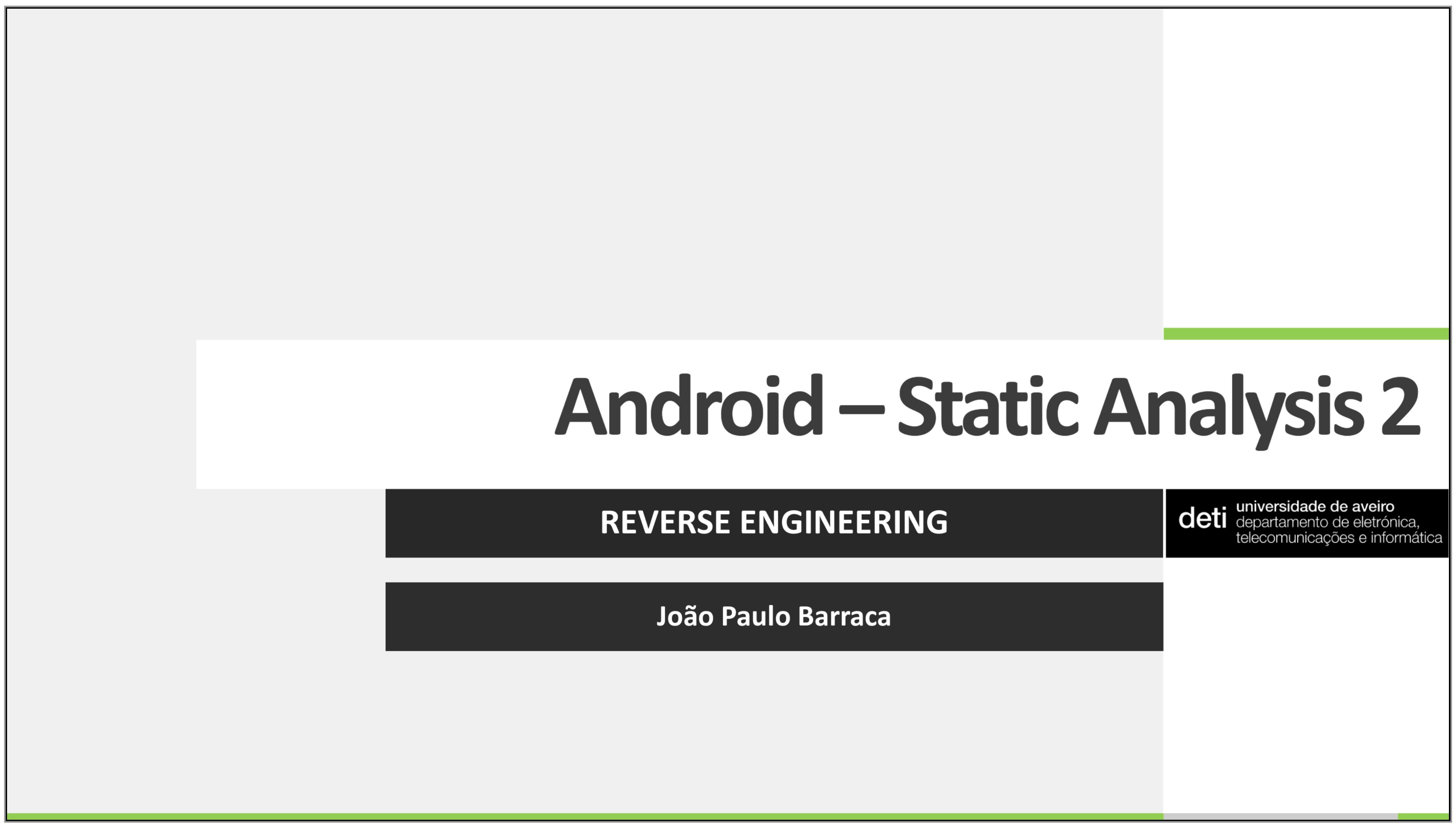
Android Binary and Web Apps

Lecture Notes

Analyzing Android applications containing JNI loaded objects, and well as applications created with Web frameworks

Previous Next



Android – Static Analysis 2

REVERSE ENGINEERING

João Paulo Barraca

deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

Download here

Practical Tasks

Exercise 1

For this exercise we will be determining the function mapping used in the `Mediacode.apk` library found here.

Create a project with Ghidra and load the binary. Any architecture will be adequate, but probably `ARM` will be the most adequate. Because the Ghidra decompiler behaves differently depending on the origin instruction set, some architectures may provide better results.

The first step is determining what type of linking is being done in the Android application.

Because we are dealing with JNI, with dynamic linking, the native methods follow a strict convention: `Java_package_class_method`. With static linking the native methods are defined on the `JNI_Onload` method (or another sub method)

As a first indicator, the commands `nm -gD` and `strings` can provide a fast indication regarding the type of linking implemented. `nm` will print the exported symbols, while `strings` will show null terminated text sequences. Use these tools and check the output to determine the linking type.

Analyzing the application `APK` with jadx will also show you which methods are native and should be found on the native library.

Then load up in Ghidra and check what is being done in the `JNI_OnLoad` method. Look at the symbols exported by the program and the execution flow in the `JNI_OnLoad` method.

Q: Can you determine the mapping, and which function is actually called?

Remember that the mapping is done through a structure with a 3 fields.

```
typedef struct {
    char *name;
    char *signature;
    void *fnPtr;
} JNINativeMethod;
```

Also remember that all native `JNI` methods have as the first argument a `JNIEnv*`. This argument contains pointers to a large number of Android methods, which will greatly improve the analysis of the library.

If you load the jni_all.gdt file into the Data Type Manager in [Ghidra[(#tools), and if you define a variable as of type `JNIEnv*` (CTRL-L), the method names will be much clearer.

Q: After finding the method, can you have an idea of what it does? Does it need to be in a native library or could it be a Java function?

Exercise 2

We have another app which may be doing something nasty with premium SMS… again. But this time it may be a little more elaborate.

Check the PhotoView.apk file and determine if it is sending SMS using static analysis. If it is, is the behavior suspect or the application only sends when the user requests it?

Apply a methodology similar to the one used previously. Write up your discoveries and share your results with the class.

Exercise 3

A company produces an Alarm system and is using a strange cipher to encrypt their messages between the Mobile Application and the Alarm Panel. Find a version of the application here. The same company provides many other applications with the functionality. It may be interesting for reversing this application to find them.

The algorithm is not an access control barrier, but protection so that others do not sniff the login and password. That would be a disaster. This exercise does not intend for you to break the cipher, but is a simple example of recognizance.

We know that it isn't exactly what it seems to be. Also, the developers shouldn't had to create a new algorithm by themselves. They should have used something… But what?

Q: Can you find the **actual** algorithm used? Share your results with the class.

Exercise 4

Analyze the Web applications available. Specifically the ionfits, AveiroApp and WeatherApp. They are respectively implemented in Ionic, ReactNative and Flutter.

Explore what information can be extracted from them.

Q: Can they be re-engineered?

To process ReactNative, check the ReactNative Decompiler and to process Flutter, check Doldrums and darter. An interesting analysis of the Flutter system, from a reverse engineering perspective, is present here.

For Flutter also check these blog posts 1,2

Write your progress, dead ends and conclusions. Be aware the in some cases, not much can be obtained and even when using pure web based technologies, the complexity of the resulting set of `HTML`, `JS` and `CSS` may be much higher than a Java application.

Tools

- nm: https://man7.org/linux/man-pages/man1/nm.1.html
- strings: https://man7.org/linux/man-pages/man1/strings.1.html
- Ghidra: https://ghidra-sre.org/
- jni_all.gdt: https://github.com/Areizen/ghidra_utils/blob/master/JNIAnalyzer/src/main/resources/jni_all.gdt
- JNIAnalyzer: https://github.com/Areizen/ghidra_utils/tree/master/JNIAnalyzer
- jadx: https://github.com/skylot/jadx
- ReactNative Decompiler: https://github.com/richardfuca/react-native-decompiler
- Doldrums: https://github.com/rscloura/Doldrums
- Darter: https://github.com/mildsunrise/darter

PREVIOUS
Android Static Analysis

NEXT
Project 1 - Android Reversing

Last updated on 28 Feb 2024