

Android Root Detection Evasion

Roberto Castellotti

Supervised by: Prof. Giovanni Lagorio

Università di Genova

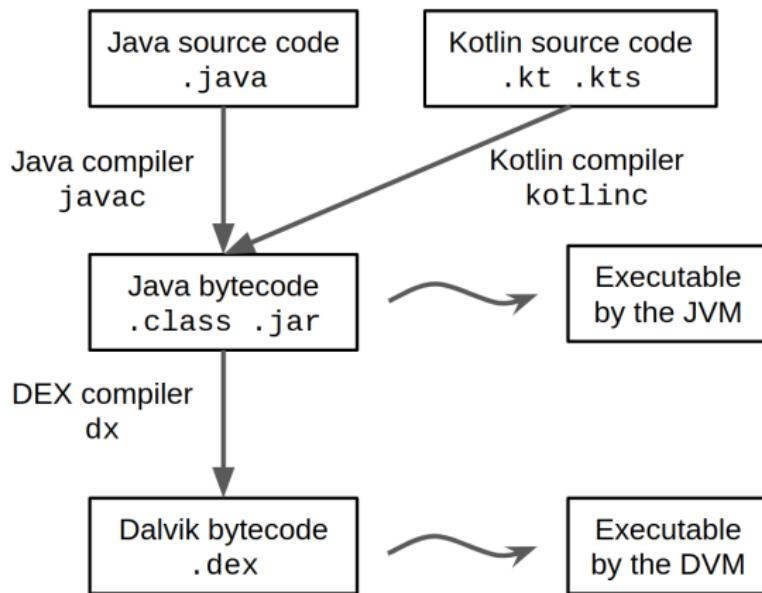
2022

fundamentals: Android



<https://source.android.com/devices/architecture>

fundamentals: java to bytecode and back



MOBISEC 2020 - 10 - Native Code

root detection: how

- check for root management apps (com.topjohnwu.magisk)

```
import android.content.pm.PackageManager;
import android.content.Context;
private final Context mContext;
PackageManager pm = mContext.getPackageManager();
pm.getPackageInfo(packageName, 0);
```

root detection: how

- check for root management apps (com.topjohnwu.magisk)

```
import android.content.pm.PackageManager;
import android.content.Context;
private final Context mContext;
PackageManager pm = mContext.getPackageManager();
pm.getPackageInfo(packageName, 0);
```

- check root cloaking appps

(com.devadvance.rootcloak, de.robv.android.xposed.installer)

root detection: how

- check for root management apps (com.topjohnwu.magisk)

```
import android.content.pm.PackageManager;
import android.content.Context;
private final Context mContext;
PackageManager pm = mContext.getPackageManager();
pm.getPackageInfo(packageName, 0);
```

- check root cloaking appps
(com.devadvance.rootcloak, de.robv.android.xposed.installer)
- check for dangerous applications (com.dimonvideo.luckypatcher)

root detection: how

- check for root management apps (com.topjohnwu.magisk)

```
import android.content.pm.PackageManager;
import android.content.Context;
private final Context mContext;
PackageManager pm = mContext.getPackageManager();
pm.getPackageInfo(packageName, 0);
```

- check root cloaking appps

(com.devadvance.rootcloak, de.robv.android.xposed.installer)

- check for dangerous applications (com.dimonvideo.luckypatcher)

- check for binaries (busybox, su)

```
File f = new File(path, filename);
boolean fileExists = f.exists();
```

root detection: how

- check for root management apps (com.topjohnwu.magisk)

```
import android.content.pm.PackageManager;
import android.content.Context;
private final Context mContext;
PackageManager pm = mContext.getPackageManager();
pm.getPackageInfo(packageName, 0);
```

- check root cloaking appps

```
(com.devadvance.rootcloak, de.robv.android.xposed.installer)
```

- check for dangerous applications (com.dimonvideo.luckypatcher)

- check for binaries (busybox, su)

```
File f = new File(path, filename);
boolean fileExists = f.exists();
```

- check if some paths are writable (/system, /sbin, /etc)

```
InputStream inputStream = Runtime.getRuntime().exec("mount")
```

patch-and-reinstall

- ➊ pull the apk

patch-and-reinstall

- ① pull the apk
- ② apktool d app.apk

patch-and-reinstall

- ① pull the apk
- ② apktool d app.apk
- ③ patch

patch-and-reinstall

- ① pull the apk
- ② apktool d app.apk
- ③ patch
- ④ apktool b app

patch-and-reinstall

- ① pull the apk
- ② apktool d app.apk
- ③ patch
- ④ apktool b app
- ⑤ sign the apk with apksigner

patch-and-reinstall

- ① pull the apk
- ② apktool d app.apk
- ③ patch
- ④ apktool b app
- ⑤ sign the apk with apksigner
- ⑥ reinstall the patched apk

patch-and-reinstall

- ➊ pull the apk
- ➋ apktool d app.apk
- ➌ patch
- ➍ apktool b app
- ➎ sign the apk with apksigner
- ➏ reinstall the patched apk
- ➐ this approach does not require the android device to be rooted

patch-and-reinstall: an example

The patching process is usually a matter of finding the methods performing root detection and patching them, here is an example from a popular financial services company's application:

```
Author: rcastellotti <me@rcastellotti.dev> 2022-06-29 01:17:32
Committer: rcastellotti <me@rcastellotti.dev> 2022-06-29 01:17:32
Parent: 4e6c812897a3d9e1187e9fafd77619b8f2b0540a (decompiled)
Child: 20844736e60bf98792250e4d4ed48af41b143 (added [REDACTED] patch)
Branches: main, remotes/origin/main
Follows:
Precedes:

i cant really believe this is the patch

[REDACTED] /smali_classes2/com/[REDACTED] /util/WuRootUtil.smali
index 6460600e0..866aaef619 100644
@@ -329,7 +329,7 @@
 
 :cond_1
 :goto_0
- const/4 v0, 0x1
+ const/4 v0, 0x0

 :goto_1
 return v0
```

Figure: patching `.method public static isDeviceRooted()Z`

frida: a dynamic toolkit

- provides ability to inject scripts into black box processes.

frida: a dynamic toolkit

- provides ability to inject scripts into black box processes.
- portable (Windows, macOS, GNU/Linux, iOS, Android)

frida: a dynamic toolkit

- provides ability to inject scripts into black box processes.
- portable (Windows, macOS, GNU/Linux, iOS, Android)
- **injected mode:** frida-server: frida-core over TCP

frida: a dynamic toolkit

- provides ability to inject scripts into black box processes.
- portable (Windows, macOS, GNU/Linux, iOS, Android)
- **injected mode:** frida-server: frida-core over TCP
- frida-core: a layer that packages up GumJS into a shared library that it injects into existing software, and provides a two-way communication channel for talking to your scripts.

frida: a dynamic toolkit

- provides ability to inject scripts into black box processes.
- portable (Windows, macOS, GNU/Linux, iOS, Android)
- **injected mode:** frida-server: frida-core over TCP
- frida-core: a layer that packages up GumJS into a shared library that it injects into existing software, and provides a two-way communication channel for talking to your scripts.
- **on device:** ./frida-server-15.1.27-android-arm64

frida: a dynamic toolkit

- provides ability to inject scripts into black box processes.
- portable (Windows, macOS, GNU/Linux, iOS, Android)
- **injected mode:** frida-server: frida-core over TCP
- frida-core: a layer that packages up GumJS into a shared library that it injects into existing software, and provides a two-way communication channel for talking to your scripts.
- **on device:** ./frida-server-15.1.27-android-arm64
- **on computer:** frida -U -l script.js -f com.package.name

frida: a dynamic toolkit

- provides ability to inject scripts into black box processes.
- portable (Windows, macOS, GNU/Linux, iOS, Android)
- **injected mode:** frida-server: frida-core over TCP
- frida-core: a layer that packages up GumJS into a shared library that it injects into existing software, and provides a two-way communication channel for talking to your scripts.
- **on device:** ./frida-server-15.1.27-android-arm64
- **on computer:** frida -U -l script.js -f com.package.name
- this approach could be better since it allows to patch applications without losing original package signature

frida: script.js

```
Java.perform(function () {
    var PackageManager = Java.use("android.app.ApplicationPackageManager");
    var NativeFile = Java.use("java.io.File");
    var Runtime = Java.use("java.lang.Runtime");

    PackageManager.getPackageInfo.overload(
        "java.lang.String",
        "int"
    ).implementation = function (pname, flags) {
        send("Bypass root check for package: " + pname);
        pname = "set.package.name.to.a.fake.one.so.we.can.bypass.it";
    }
    return this.getPackageInfo
        .overload("java.lang.String", "int")
        .call(this, pname, flags);
};

NativeFile.exists.implementation = function () {
    var name = NativeFile.getName.call(this);
    send("Bypass return value for binary: " + name);
    return false;
};
});
```

partially adapted from [@dzoncerzy/fridantiroot](#)

com.revolut.revolut (fintech)

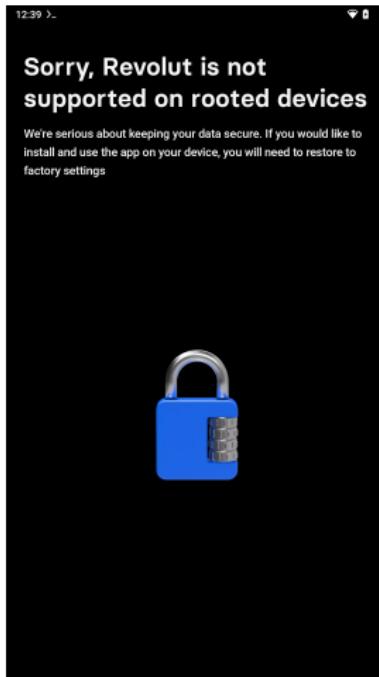


Figure: original apk

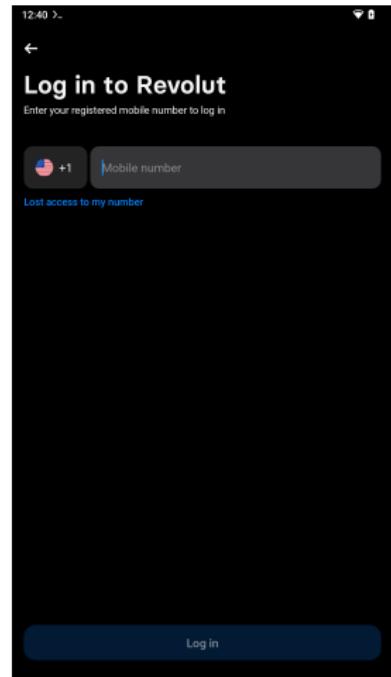


Figure: patched

com.scottyab.rootbeer.sample



Figure: original apk

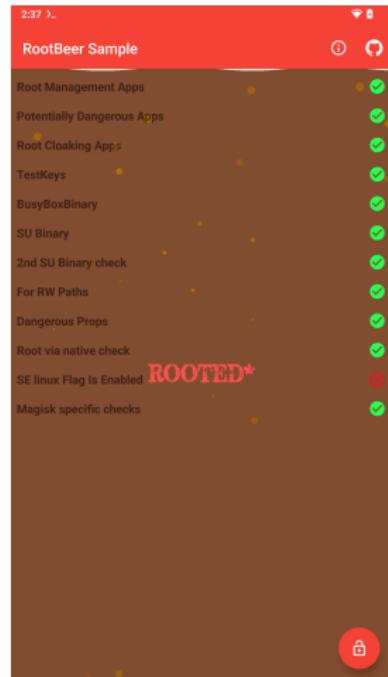


Figure: patched

a popular financial services company's application

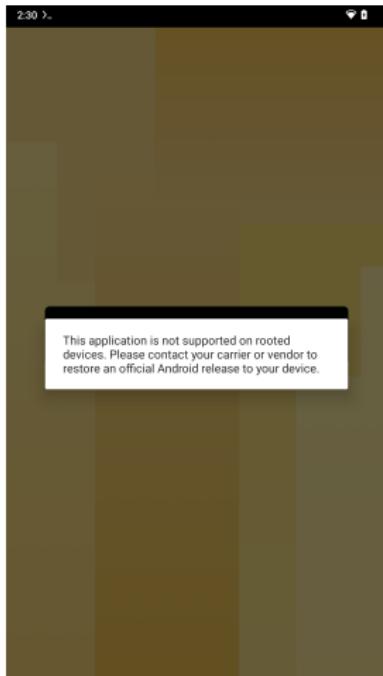


Figure: original apk

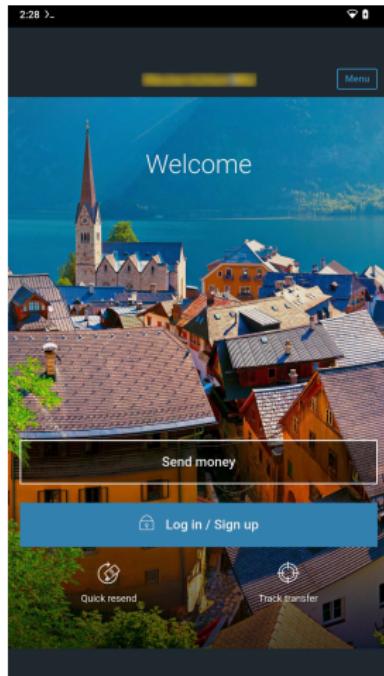


Figure: patched

tools and sources

- [Android Studio](#)
- [iBotPeaches/Apktool](#): A tool for reverse engineering Android apk files
- [skylot/jadx](#): Dex to Java decompiler
- [scottyab/rootbeer](#): root checking Android library and sample app
- <https://en.wikipedia.org/wiki/Dalvik>
- [Dalvik Bytecode](#)
- [Anroid Runtime](#)
- <https://developer.android.com/guide/components/fundamentals>