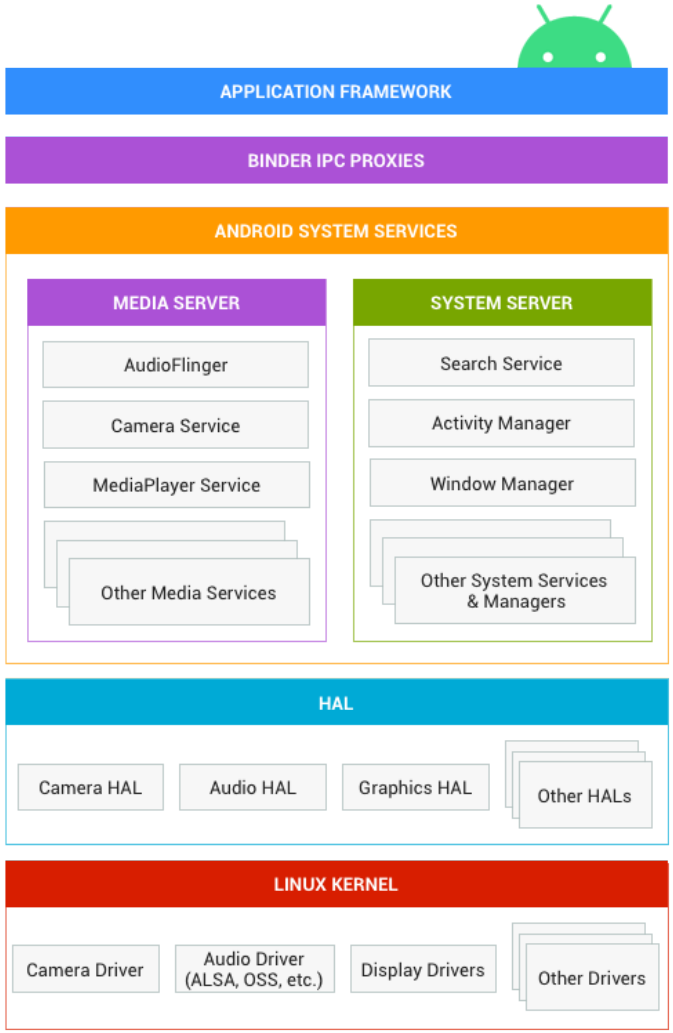
Android Root Detection Evasion

Roberto Castellotti
Supervised by: Prof. Giovanni Lagorio

Università di Genova

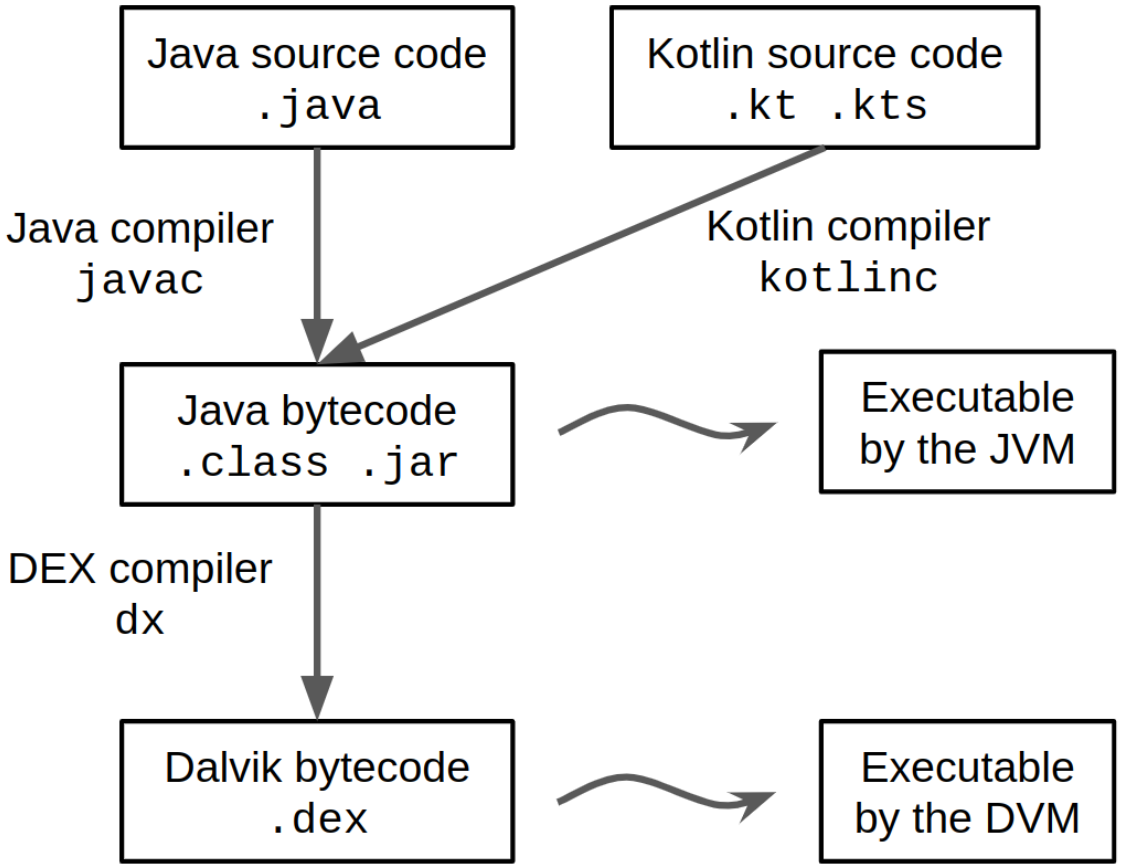
2022

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 apps (`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 apps (`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 apps (`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 apps (`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

1. 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

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
7. 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:  28844736e60b1fb98792250e4d4ed48afd41b143 (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..866aaf619 100644
@@ -329,7 +329,7 @@

     :cond_1
     :goto_0
-    const/4 v0, 0x1
+    const/4 v0, 0x0

     :goto_1
     return v0
```

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

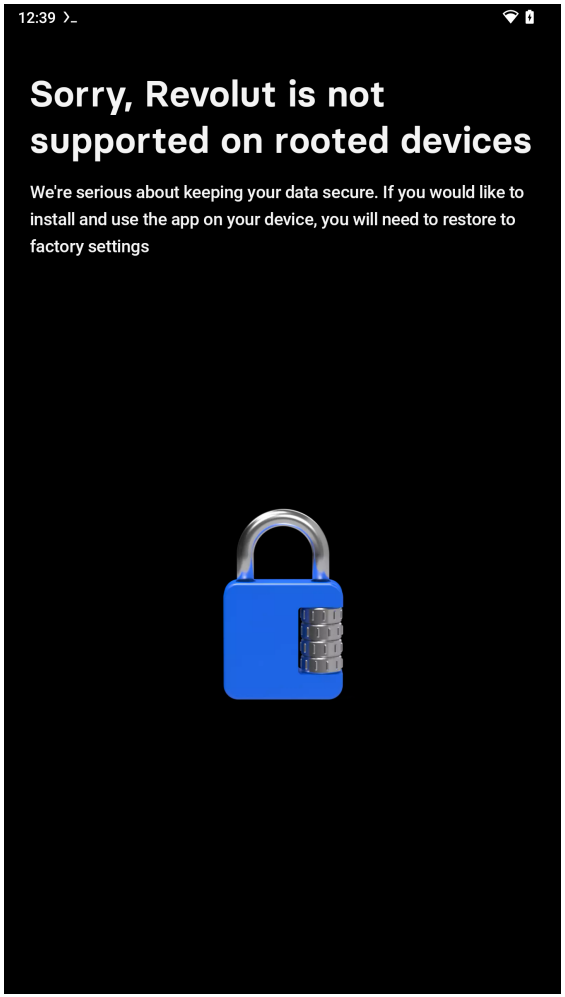
- provides ability to inject scripts into black box processes.
- portable (Windows, macOS, GNU/Linux, iOS, Android)
- **injected mode**: `frida-server`: `frida-core` over TCP
- `frida-core`: a layer that packages up GumJS into a shared library that it injects into existing software, and provides a two-way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

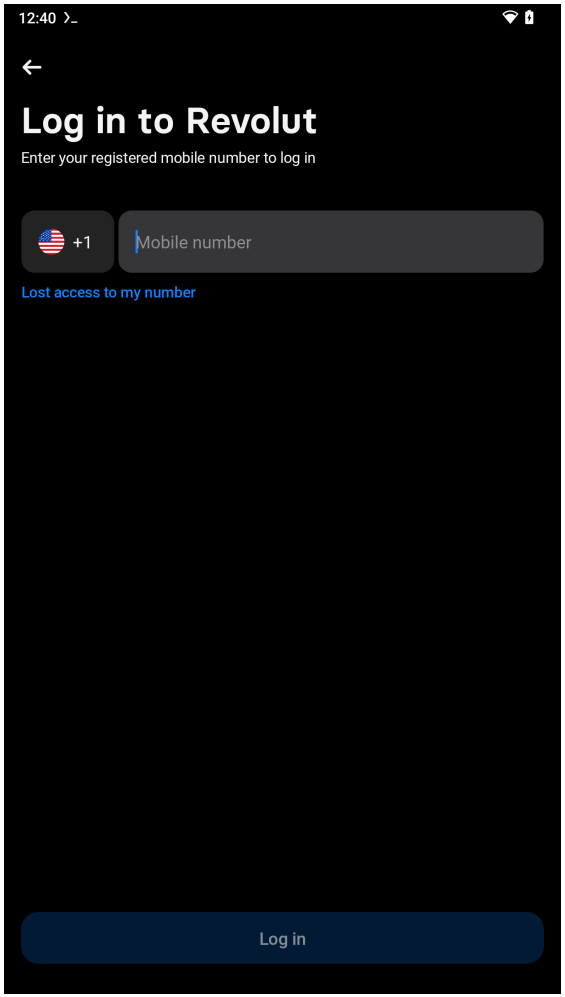
```
Java.perform(function () {
  var PackageManager = Java.use("android.app.ApplicationPackageManager");
  var NativeFile = Java.use("java.io.File");
  var Runtime = Java.use("java.lang.Runtime");

  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 @dzonerzy/fridantiroot

com.revolut.revolut (fintech)



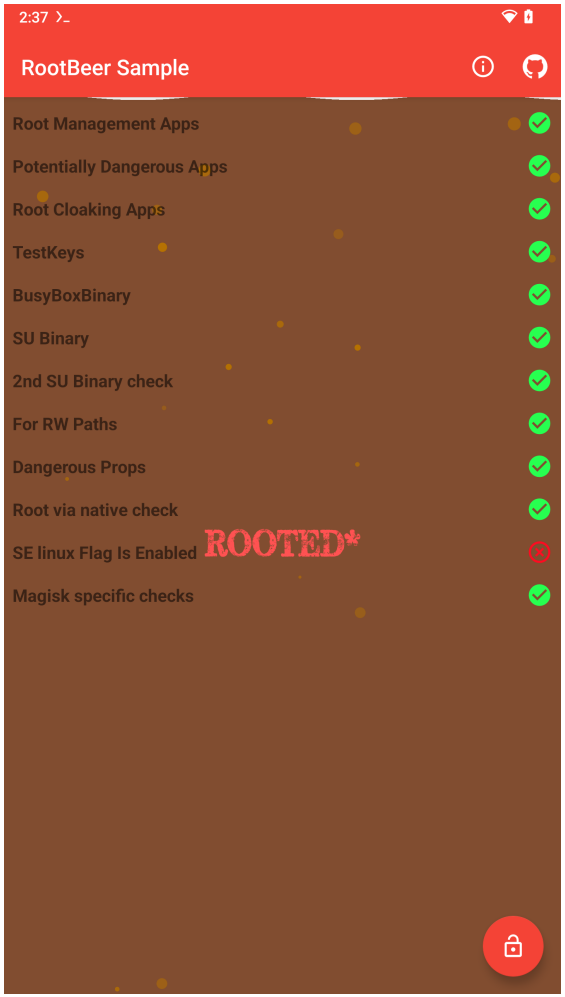
original APK



patched

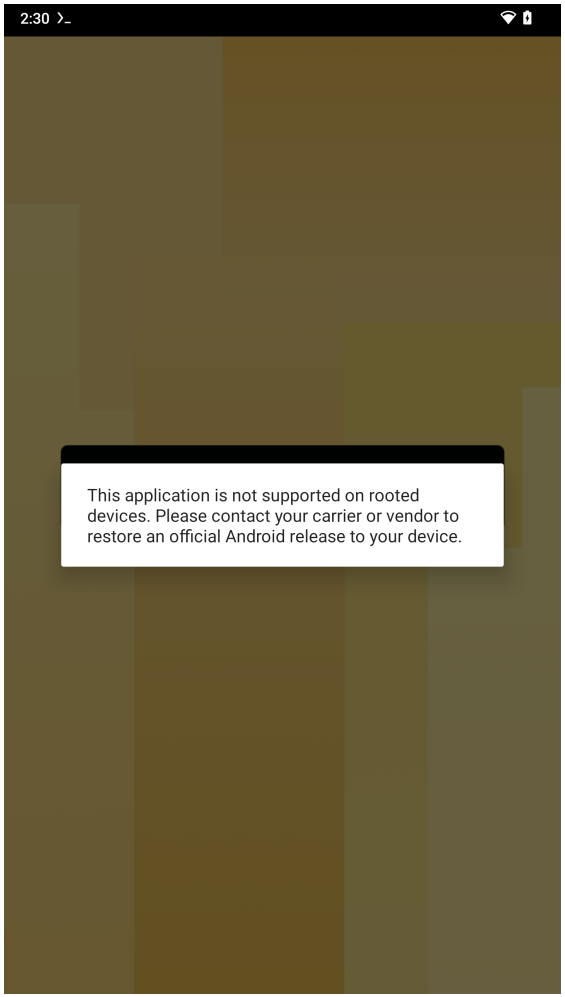
com.scottyab.rootbeer.sample

original APK

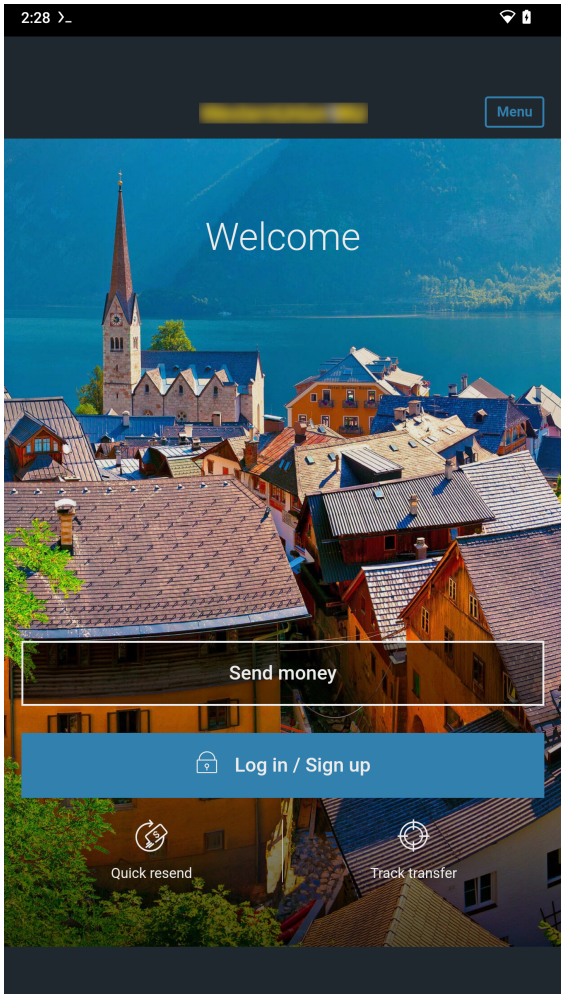


patched

a popular financial services company's application



original APK



patched

Thank You for your attention,
Questions?

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