Deal with Reverse Engineering

@catvinhquang

Sep 20th 2019

Agenda

I. ProGuard / R8: Obfuscate code
II. Signature recognition
III. Hide view IDs (like Facebook / Messenger)
IV. Extra tips

Demo: https://github.com/catvinhquang/deal-with-reverse-engineering

I. ProGuard / R8: How it works?

Android Gradle plugin 3.4.0+ no longer uses ProGuard to perform compile-time code optimization. Instead, the plugin works with the R8 compiler to handle the following compile-time tasks:

- Code shrinking: removes unused classes, fields, methods, and attributes
- Resource shrinking: removes unused resources
- Obfuscation: shortens the name of classes and members
- Optimization: inspects and rewrites your code

I. ProGuard / R8: Setup

```
android {
    buildTypes {
        release {
            // Enables code shrinking, obfuscation, and optimization for only
            // your project's release build type.
            minifyEnabled true

            // Enables resource shrinking, which is performed by the
            // Android Gradle plugin.
            shrinkResources true

            // Includes the default ProGuard rules files that are packaged with
            // the Android Gradle plugin. To learn more, go to the section about
            // R8 configuration files.
            proguardFiles getDefaultProguardFile(
                    'proguard-android-optimize.txt'),
                    'proguard-rules.pro'
        }
    }
    ...
}
```

build.gradle in app module

II. Signature recognition: Signing process

II. Signature recognition: Detect APK changes

Figure 1. Local check

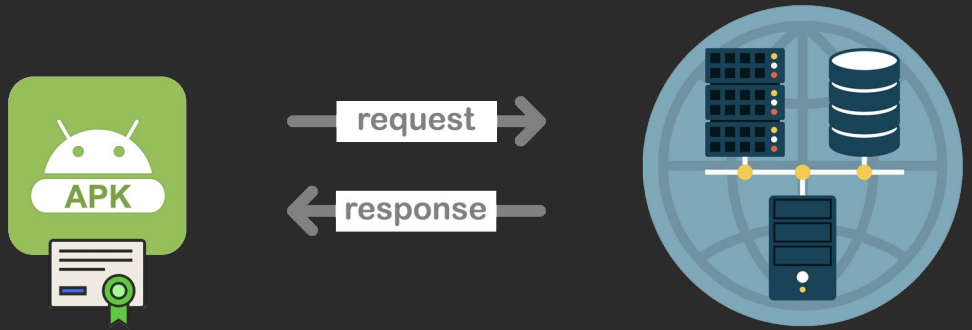


Figure 2. Server check

III. Hide view IDs: Why?

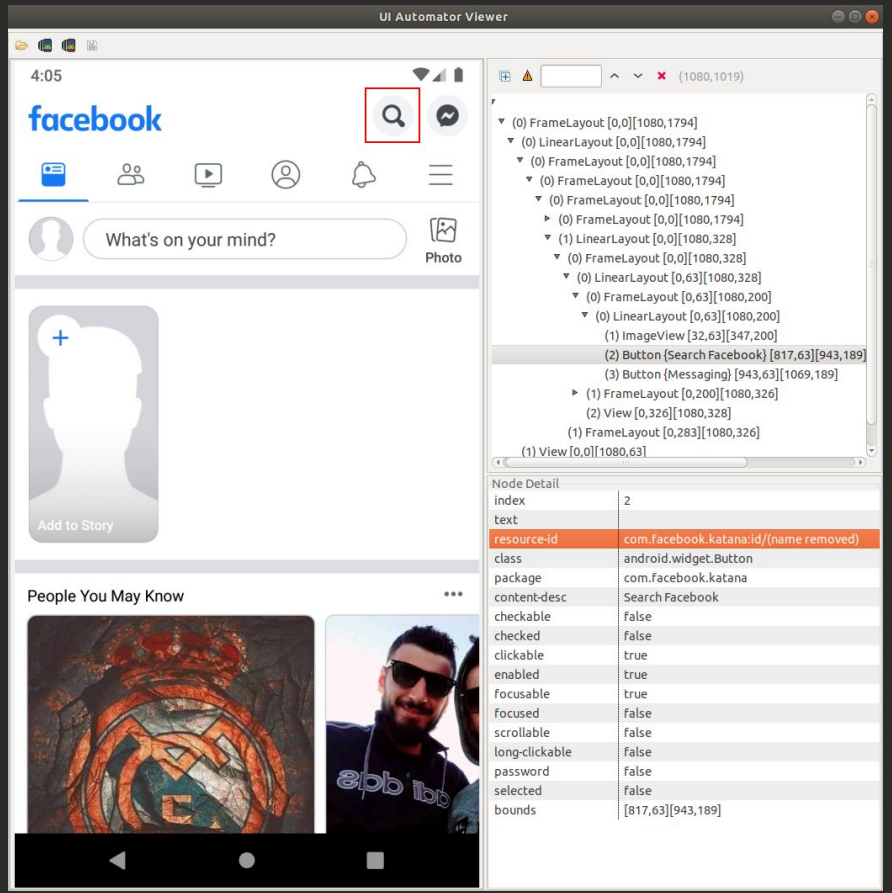
Tools to find view IDs:

- androidsdk / tools / bin / **uiautomatorviewer**
- androidsdk / tools / platform-tools / **adb shell dumpsys activity top**

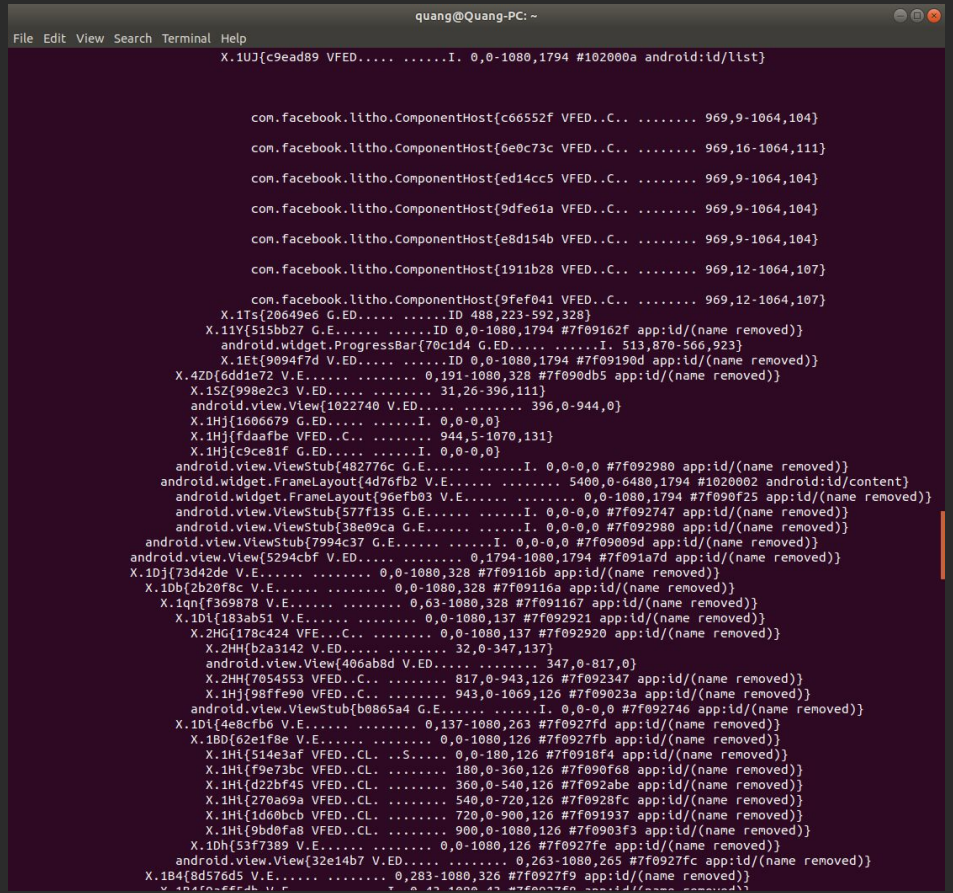
The effect of hiding view IDs:

- Positive: Determining the point of attack will be harder
- Negative: Automated testing will be harder

uiautomatorviewer



adb shell dumpsys activity top



III. Hide view IDs: Implementation

- Method 1: View wrapper

```
public class MyTextView extends TextView {

    public MyTextView(Context context) {
        super(context);
    }

    // works on uiautomatorviewer
    @Override
    public void onInitializeAccessibilityNodeInfo(AccessibilityNodeInfo info) {
        super.onInitializeAccessibilityNodeInfo(info);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR2) {
            info.setViewIdResourceName("(name removed)");
        }
    }

}
```

III. Hide view IDs: Implementation

- Method 2: Resources wrapper

```
public class CustomResources extends Resources {

    public CustomResources(Resources res) {
        super(res.getAssets(), res.getDisplayMetrics(), res.getConfiguration());
    }

    // works on uiautomatorviewer
    @Override
    public String getResourceName(int resid) throws NotFoundException {
        return "(name removed)";
    }

    // works on "adb shell dumpsys activity top"
    @Override
    public String getResourceEntryName(int resid) throws NotFoundException {
        return "(name removed)";
    }

}
```

IV. Extra tips

- Dynamic code
- Detect rooted device
- Detect rom debug
- Encrypt resources
- Store important resources on server

References

https://developer.android.com/studio/build/shrink-code

https://android-developers.googleblog.com/2018/11/r8-new-code-shrinker-from-google-is.html

https://www.guardsquare.com/en/products

https://www.guardsquare.com/en/products/proguard/manual/usage

https://www.guardsquare.com/en/blog/proguard-and-r8

https://source.android.com/setup/build/building

Thank you for listening!