Google Play Protect

Thinking outside of the (sand)box

Łukasz Siewierski, Botconf 2017

Google Play
Protect

Security features in Android

- SELinux
- Application sandbox
- Permission model
- Verified Boot
- ASLR
- TEE

Security features in Android

- SELinux
- Application sandbox
- Permission model
- Verified Boot
- ASLR
- TEE

Agenda

1. What is application sandbox?
2. How do malicious apps try to circumvent or break it?
3. Can you create your own sandbox?
4. What steps are we taking to reduce risks for the user?

Google Play
Protect

Why do we need Android sandbox?

1. Prevents spyware from accessing other app’s data
2. Prevents apps from posing as other apps (or uids)
3. Makes it easy to attribute actions to specific apps (or uids)
4. Allows for strict permission enforcement

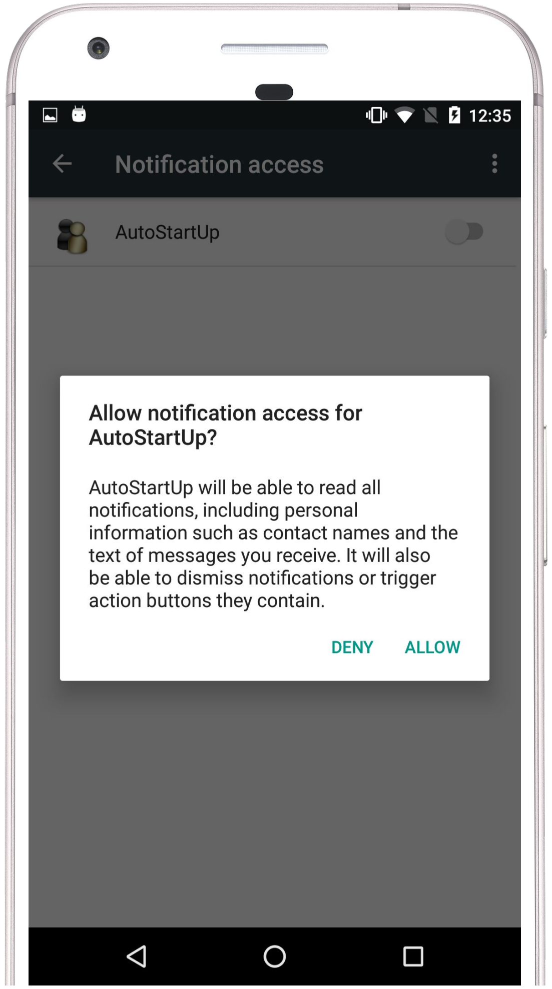
… and all of this bothers malware authors!

How to break out of the sandbox?

Google Play
Protect

Method 0.5 (not really creative): social engineering

1. Ask users for a number of really excessive and hard to grant permissions.
2. Cross your fingers they will be able to do that.
3. Profit (??)



Allow notification access for AutoStartUp?

AutoStartUp will be able to read all notifications, including personal information such as contact names and the text of messages you receive. It will also be able to dismiss notifications or trigger action buttons they contain.

Method 0.5 (not really creative): social engineering

```
if (packageName.equals("com.instagram.android")) {
  info = social.InstagramDetect.igramOnPosted(mycontext, p12);
}

if (packageName.equals("com.viber.voip")) {
  info = social.ViverDetect.viberOnPosted(mycontext, p12);
}

if (packageName.equals("com.facebook.katana")) {
  info = social.FbMsgDetect.FbMsgOnPosted(mycontext, p12);
}
```

Method 0.5 (not really creative): social engineering

Problems with this approach:

- Doesn’t really break a sandbox
- Requires a lot of luck
- User is clearly warned

Method I: user will help me...

1. Exploit the phone and install Xposed.
2. Hide malware within an Xposed module.
3. Profit!

What is Xposed?

“Xposed is a framework for modules that can change the behavior of the system and apps without touching any APKs.”

```
findAndHookMethod("com.android.systemui.statusbar.policy.Clock", lpparam.classLoader, "updateClock",
new XC_MethodHook() {
  @Override
  protected void beforeHookedMethod(MethodHookParam param) throws Throwable {

    // this will be called before the clock was updated by the original method

  }
  @Override
  protected void afterHookedMethod(MethodHookParam param) throws Throwable {

    // this will be called after the clock was updated by the original method

  });
```

Google Play
Protect

Why is it used?

Prevents malicious app from showing on the list of installed apps.

```
findAndHookMethod("android.app.ApplicationPackageManager", lpparam.classLoader, "getInstalledApplications",
new XC_MethodHook() {

  @Override
  protected void afterHookedMethod(MethodHookParam param) throws Throwable {
    java.util.List result = param.getResult();
    …
    if (list_element.packageName.equals("my.malicious.app"))) {
      result.remove(list_element);
    }
    …
    return result;

  });
```

Why is it used?

Give malicious app every permission.

```
findAndHookMethod("android.app.ContextImpl", lpparam.classLoader, "checkPermission", new
XC_MethodHook() {

  @Override
  protected void afterHookedMethod(MethodHookParam param) throws Throwable {

    …
    if (param.equals("my.malicious.app"))) {
      p4.setResult(Integer.valueOf(0));
    }
    …
    return result;

  });
```

Google Play
Protect

Method I: user will help me...

Problem with this approach:

- Requires user to actively seek out and install Xposed
- Requires an exploit

Method II: let’s break everything first!

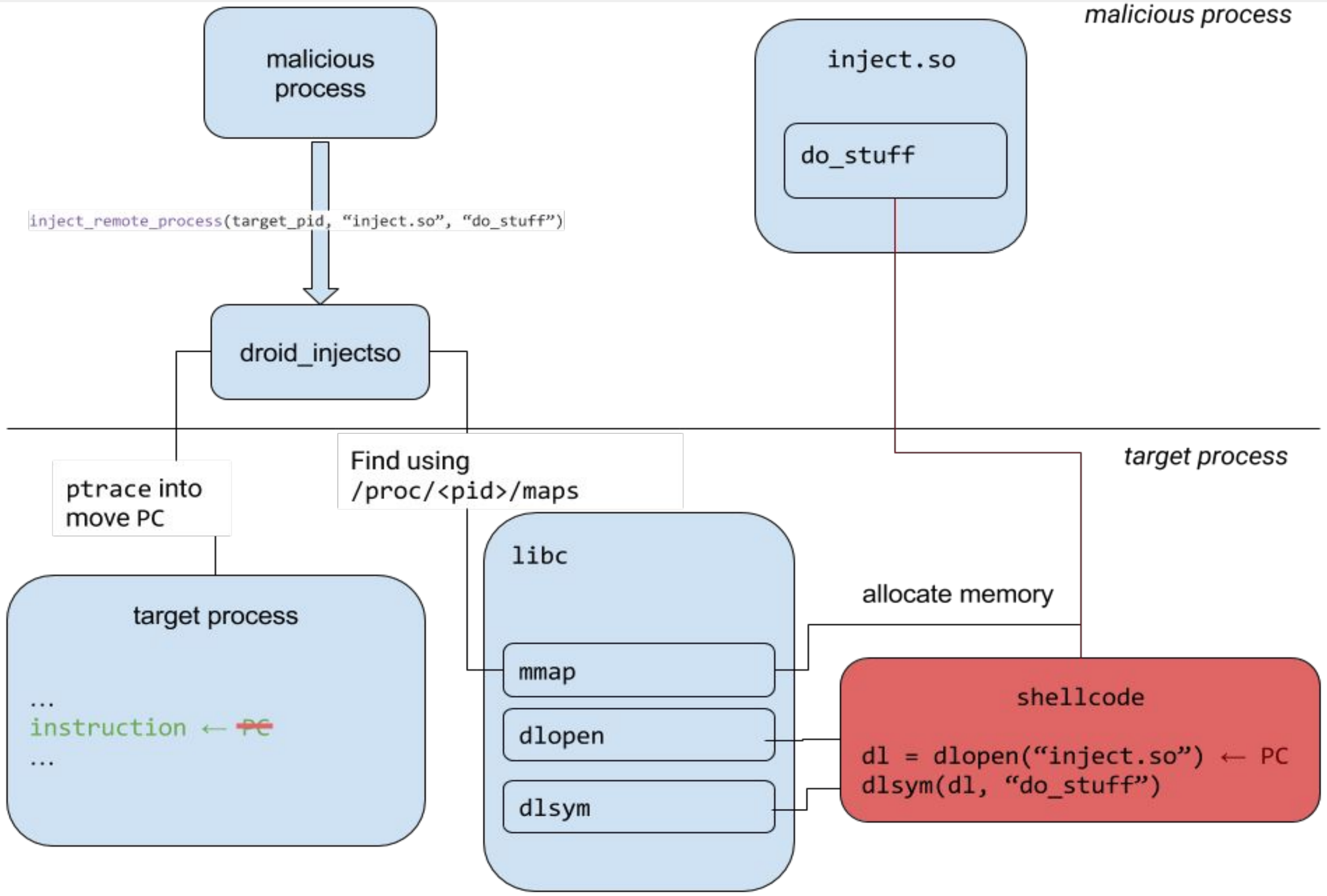
1. Exploit the phone and gain root.
2. Inject code into other processes.
3. Profit!

How some apps inject code

There are currently three popular frameworks for Android code injection:

- Android Dynamic Binary Instrumentation Toolkit (`adbi`)
- `injectDemo`
- `droid_injectso`

Code injection general workflow



Google Play Protect

droid_injectso usage example: SMS sending trojan

```
java.io.File bin_directory = context.getDir("bin", 0);
String libvangd = new
StringBuilder(String.valueOf(bin_directory.getAbsolutePath())).append("/").append("libvangd.so").toStri
ng();
int com_android_phone_pid = com.mtsoft.bosonsdk.e.d(context);
if (com_android_phone_pid != 0) {
  Object[] command_parts = new Object[4];
  command_parts[0] = new java.io.File(new
StringBuilder(String.valueOf(bin_directory.getAbsolutePath())).append("/injectso").toString()).getAbsol
utePath();
  command_parts[1] = Integer.valueOf(com_android_phone_pid);
  command_parts[2] = libvangd;
  command_parts[3] = "/data/local/tmp-drp.apk@com.boson.drop.MainClass@test";
  com.mtsoft.bosonsdk.l.execute("/system/bin/.nbwayxwzt", String.format("%s %d %s %s", command_parts));
```

Google Play
Protect

droid_injectso usage example: SMS sending trojan

```
java.io.File bin_directory = context.getDir("bin", 0);
String libvangd = new
StringBuilder(String.valueOf(bin_directory.getAbsolutePath())).append("/").append("libvangd.so").toStri
ng();
int com_android_phone_pid = com.mtsoft.bosonsdk.e.d(context);
if (com_android_phone_pid != 0) {
  Object[] command_parts = new Object[4];
  command_parts[0] = new java.io.File(new
StringBuilder(String.valueOf(bin_directory.getAbsolutePath())).append("/injectso").toString()).getAbsol
utePath();
  command_parts[1] = Integer.valueOf(com_android_phone_pid);
  command_parts[2] = libvangd;
  command_parts[3] = "/data/local/tmp-drp.apk@com.boson.drop.MainClass@test";
  com.mtsoft.bosonsdk.l.execute("/system/bin/.nbwayxwzt", String.format("%s %d %s %s", command_parts));
```

Google Play
Protect

droid_injectso usage example: SMS sending trojan

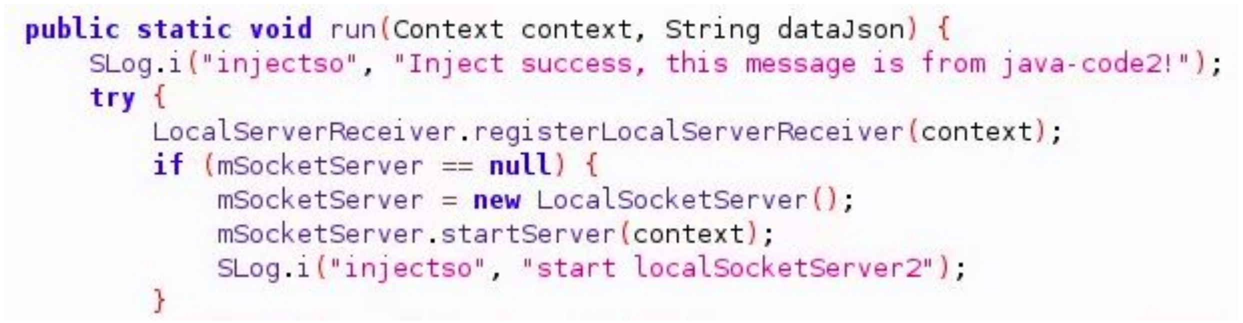
```
su injectso <com.android.phone PID> libvangd tmp-drp.apk@com.boson.drop.MainClass@test
```

or, in English:

- Run `injectso` as root
- Inject into `com.android.phone` native code from the `libavangd` library
- Wait, that last parameter shouldn’t be there though...

droid_injectso usage example: SMS sending trojan

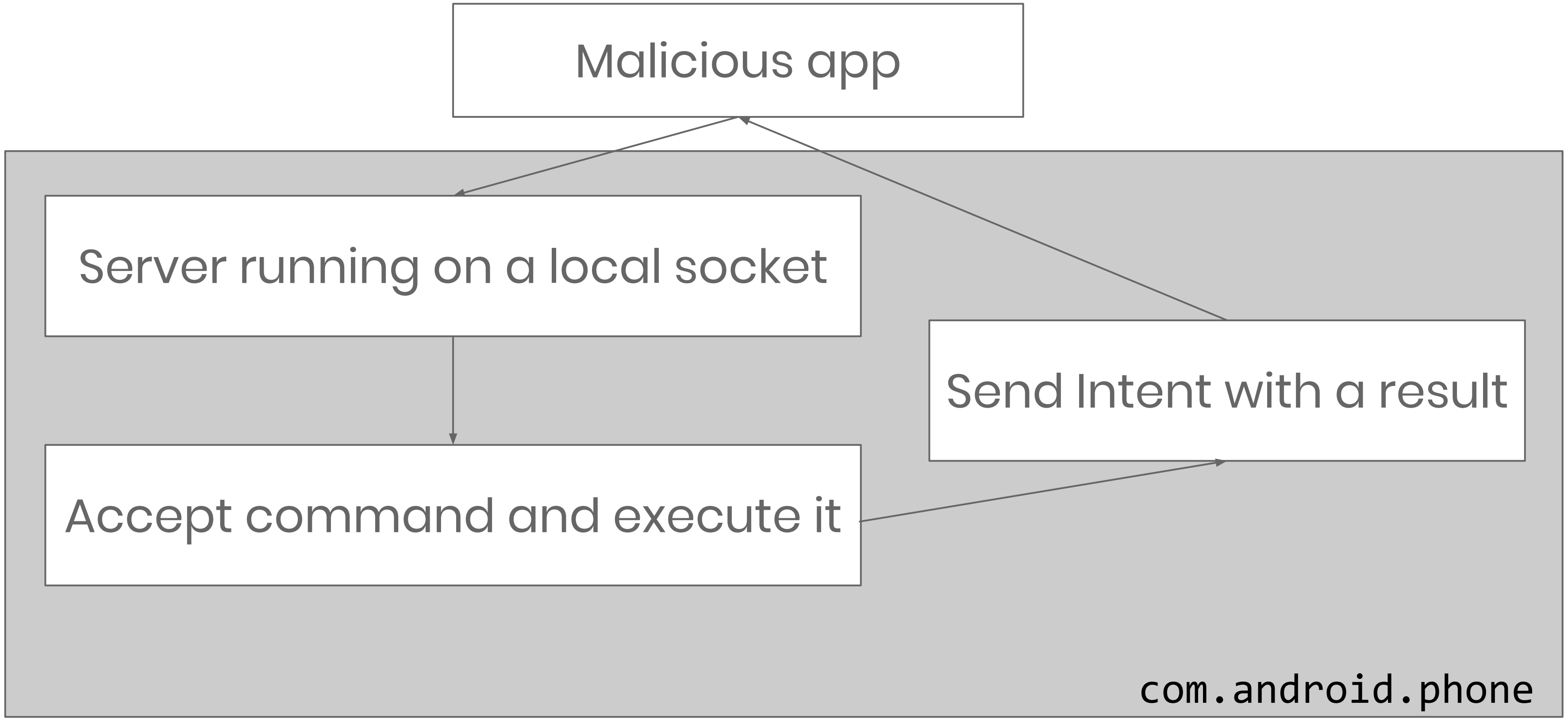
Modified droidinject_so: instead of injecting native code, it allows users to inject Dalvik code.



```
public static void run(Context context, String dataJson) {
    SLog.i("injectso", "Inject success, this message is from java-code2!");
    try {
        LocalServerReceiver.registerLocalServerReceiver(context);
        if (mSocketServer == null) {
            mSocketServer = new LocalSocketServer();
            mSocketServer.startServer(context);
            SLog.i("injectso", "start localSocketServer2");
        }
```

Google Play
Protect

droid_injectso usage example: SMS sending trojan

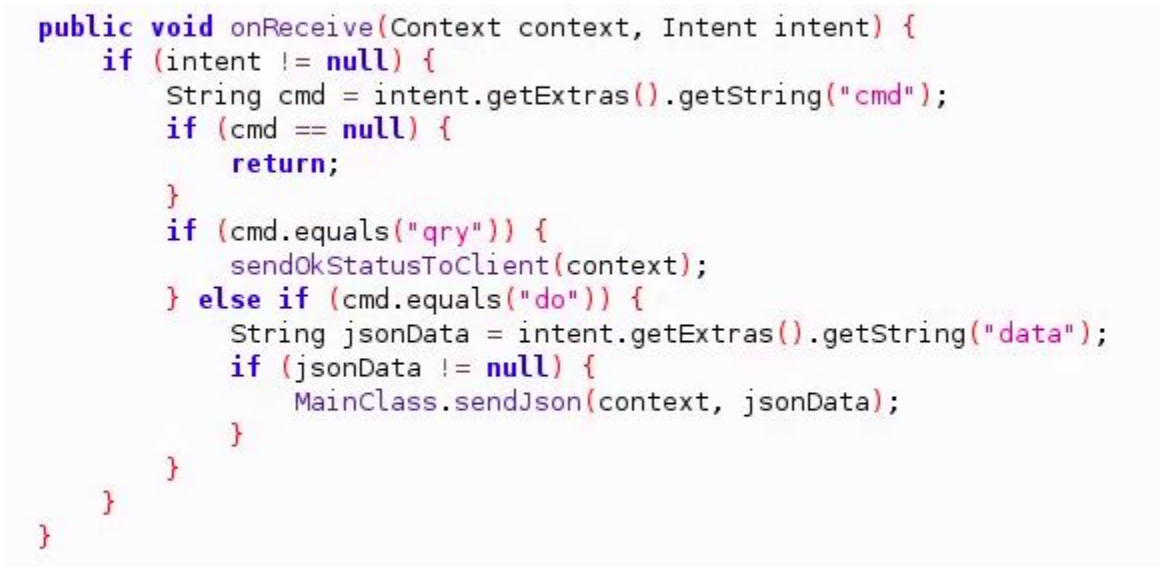

Malicious app
Server running on a local socket
Accept command and execute it
Send Intent with a result
com.android.phone

Google Play Protect

droid_injectso usage example: SMS sending trojan

Accepts two kinds of commands:

- `do` - delete a text message
- `qry` - heartbeat



```java
public void onReceive(Context context, Intent intent) {
    if (intent != null) {
        String cmd = intent.getExtras().getString("cmd");
        if (cmd == null) {
            return;
        }
        if (cmd.equals("qry")) {
            sendOkStatusToClient(context);
        } else if (cmd.equals("do")) {
            String jsonData = intent.getExtras().getString("data");
            if (jsonData != null) {
                MainClass.sendJson(context, jsonData);
            }
        }
    }
}
```

Google Play
Protect

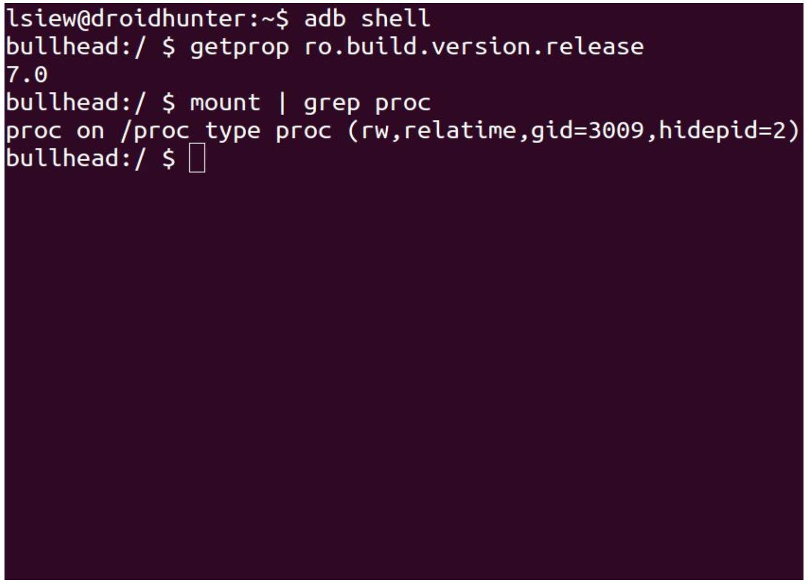
Method II: let’s break everything first!

Problems with this approach:

- Requires an exploit
- Can have unexpected side effects
- Doesn’t work out-of-the-box on newer Android versions...

Measures against code injection frameworks

`/proc/<pid>` access is disabled with `hidepid=2` mount option starting with Android Nougat



```
lsiew@droidhunter:~$ adb shell
bullhead:/ $ getprop ro.build.version.release
7.0
bullhead:/ $ mount | grep proc
proc on /proc type proc (rw,relatime,gid=3009,hidepid=2)
bullhead:/ $ 
```

Summary

- Every method of doing code injection requires an exploit or users willingly rooting their devices.
- Although it's rather rare, we see code injection techniques on Android.
- It's probably because with changes introduced in Android Nougat and new exploit mitigations introduced in new Android versions, getting a reliable exploit is very hard.

THANK YOU

Google Play Protect

Quick recap

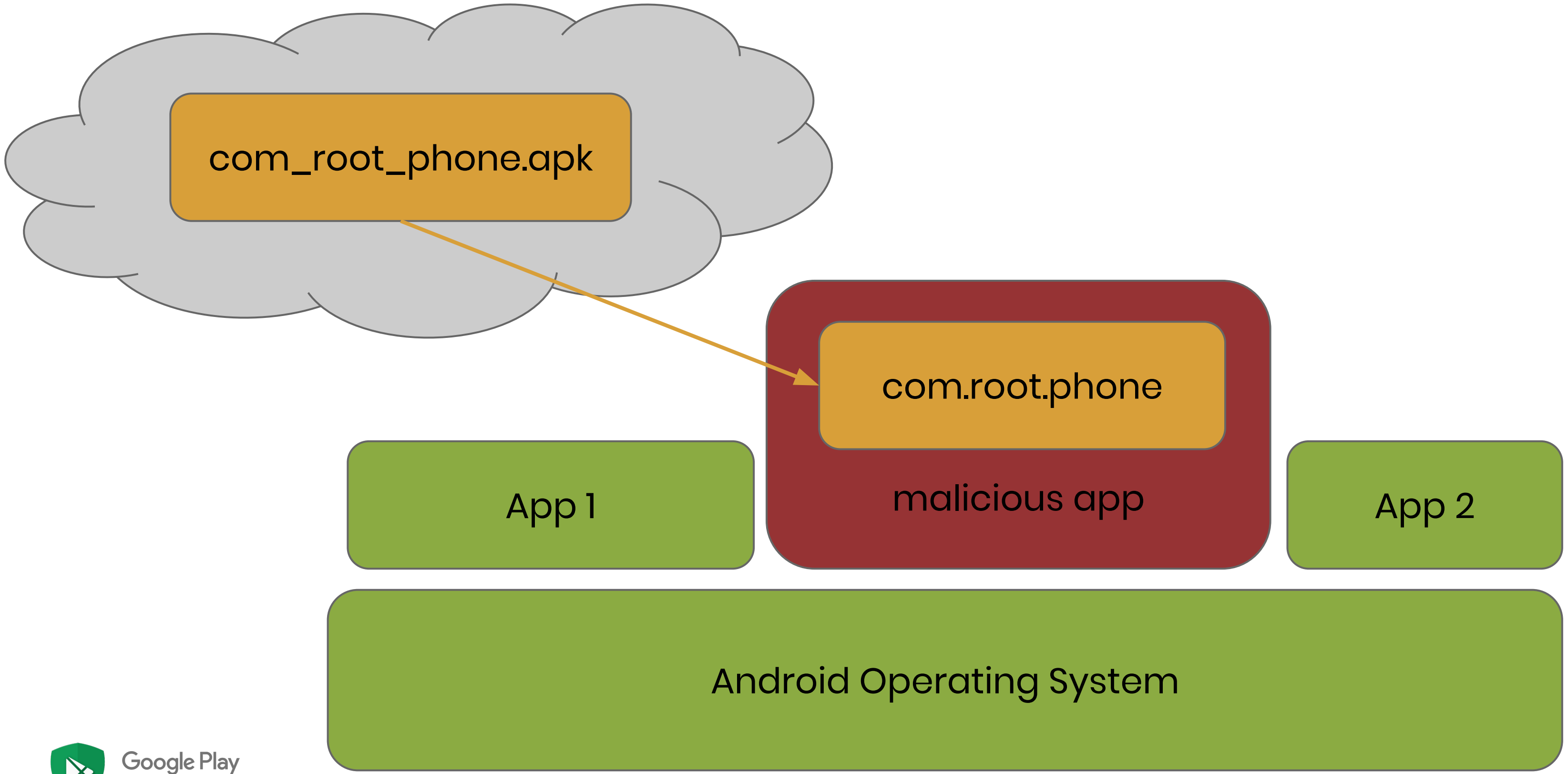
Every method of doing code injection requires an exploit or users willingly rooting their devices.

Exploits are getting harder and harder to come by…

… maybe we could use some of those popular rooting apps instead? But we would have to hide it somehow…

I’m going to make my own sandbox, with apps and API

Using Virtual App containers

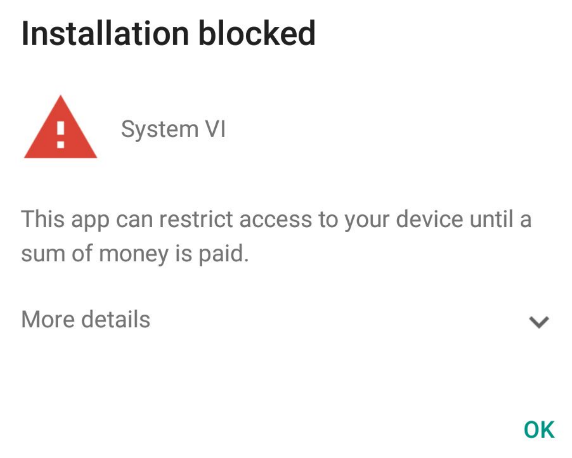


Google Play policy and Google Play Protect

The following are explicitly prohibited:

(. . .)

- Apps or SDKs that download executable code, such as dex files or native code, from a source other than Google Play.



Installation blocked

System VI

This app can restrict access to your device until a sum of money is paid.

Google Play Protect