

Hooking on Android

written by Mert SARICA | 1 January 2021

Although our topic is the Android world, when it comes to hooking, I first think of the illegal electricity that is drawn by hooking onto energy transmission lines and into homes. In the Android world, we also use a similar method when we want to dynamically analyze or intervene in applications. So why do we need this? Sometimes, when we want to do a cybersecurity research on Android applications or during a penetration test, we need to analyze the target Android application to find security vulnerabilities. To do this, we usually start by converting the target Android application into source code and doing static code analysis. However, in today's Android applications, the codes are hidden (obfuscation) and made difficult to understand, so we try to analyze the application dynamically using emulators such as Genymotion. The reason for saying "try" is that today, mobile applications like Snapchat apply many methods to prevent dynamic analysis in addition to static analysis.

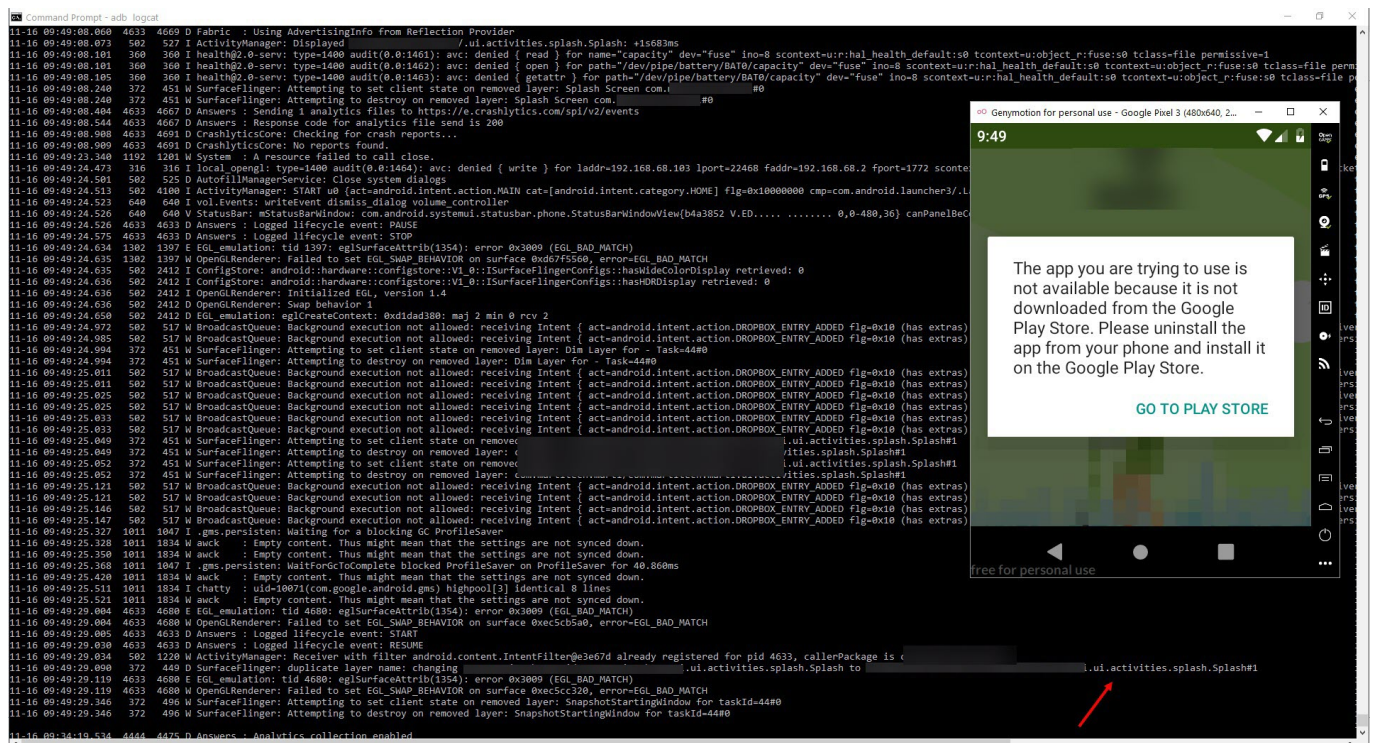
During my trip to Stockholm in December 2019, as I was exploring the city, I noticed that people were passing by me on stylish electric scooters. Since their use is extremely widespread in Europe, I remembered that some security vulnerabilities had been detected in electric scooter applications over time as they began to catch the attention of security researchers. As electric scooters and their applications are also starting to become popular in my country, I decided to take a look at one of the Android applications from a security researcher's perspective, in order to satisfy my curiosity before using them. Of course, as always, reality did not meet my expectations and due to the obstacles I faced, this blog post came about. :)

As a security researcher with limited time, I did not want to waste time with static code analysis, so I downloaded the APK file using the APK Downloader web application and loaded it into the GenyMotion emulator. After running the application, I was surprised to see a warning message that the APK file had not been downloaded from Google Play. :)

The app you are trying to use is not available because it is not downloaded from the Google Play Store. Please uninstall the app from your phone and install it on the Google Play Store.

GO TO PLAY STORE

So, before diving into blind static source code analysis, I ran the adb logcat command to view system messages on the command line and then ran the application again. I used the jadx tool to convert the APK file to source code and began examining the activities.splash.Splash file to find the function that I thought was related to the warning message that appeared on the screen. At the end of the file, I noticed the verifyInstallerId function, which immediately caught my attention. When I searched for this function on Google, I saw that it was used for the exact purpose. This function used the Android's getInstallerPackageName function to check if the application was installed by Google Play. If the application was installed by Google Play, the installerPackageName variable would have a non-zero value.



```

124 public int getContentView() {
125     return R.layout.activity_splash;
126 }
127
128
129 public Context getContext() {
130     return this;
131 }
132
133 public void initView() {
134     if (this.presenter == null) {
135         this.presenter = new SplashPresenter(this);
136     }
137     if (!verifyInstallerId(this)) {
138         showWrongAppVersion();
139         return;
140     }
141     this.presenter.getConfig();
142     setSnackBarView(FindViewById(R.id.rootlayout), true);
143     this.versionCode = 75;
144 }
145
146 public void onError(String str) {
147     if (!"inprogress".equals(str)) {
148         hideProgress();
149         if (!str.equals("") && !str.equals(Constants.EXCEPTION)) {
150             showAlert(str);
151         }
152     } else if (progressIsShown()) {
153         setProgressMessage();
154     }
155 }
156
157 public void onHasRide(boolean z) {
158     if (z) {
159         gotoActiveRide();
160     } else {
161         gotoHomePage();
162     }
163 }
164
165 public void onLoadConfig() {
166     if (LocalDataManager.getInstance().getConfig().getAndroidVersion() > this.versionCode) {
167         showUpdateAler();
168     } else {
169         checkLogin();
170     }
171 }
172
173 public boolean verifyInstallerId(Context context) {
174     ArrayList arraylist = new ArrayList(Arrays.asList(new String[]{"com.android.vending", "com.google.android.feedback"}));
175     String installerPackageName = context.getPackageManager().getInstallerPackageName(context.getPackageName());
176     return installerPackageName != null && arraylist.contains(installerPackageName);
177 }
178 }

```

JADX memory usage: 0,65 GB of 4,00 GB

android - Detect if an app is installed from play store

stackoverflow.com/questions/37539949/detect-if-an-app-is-installed-from-play-store

Stack Overflow

Detect if an app is installed from Play store

Asked 3 years, 5 months ago Active 3 months ago Viewed 6k times

I want to check and allow the use of my app just if it has been downloaded from the Play store, and it has not been shared by other user or from any other source. How can I prevent a user to use the app if it has not been downloaded from the Google Play store?

android

share improve this question

edited May 31 '16 at 10:14 **Javier S.** 293 ● 3 ● 9

asked May 31 '16 at 7:53 **Manthan Patel** 1,564 ● 6 ● 16

Possible duplicate of [How to know an application is installed from google play or side-load?](#) – Julien Lopez May 31 '16 at 8:04

add a comment

2 Answers

23

This method will check if your app has been installed from the Play Store.

```

boolean verifyInstallerId(Context context) {
    // A list with valid installers package name
    List<String> validInstallers = new ArrayList<>(Arrays.asList("com.android.vending", "com.
    // The package name of the app that has installed your app
    final String installer = context.getPackageManager().getInstallerPackageName(context.getP
    // true if your app has been downloaded from Play Store
    return installer != null && validInstallers.contains(installer);
}

```

Some days ago I released an Android library, [PiracyChecker](#), that protects your app using some techniques, such as Google Play Licensing (LVL), APK signature protection and installer ID (this one).

share improve this answer

edited May 31 '16 at 8:10 answered May 31 '16 at 8:04 **Javier S.**

Blog

- We're Rewarding the Question Askers
- Why is the Migration to Python 3 Taking So Long?

Featured on Meta

- Feedback post: Moderator review and reinstatement processes
- Post for clarifications on the updated pronouns FAQ
- New Post Notices (Closed/On Hold/etc.) rolling out on Stack Overflow
- Upvotes on questions will now be worth the same as upvotes on answers

Remote jobs

- Senior DevOps Engineer (Remote) X-Team No office location REMOTE amazon-web-services docker
- Senior Full Stack Engineer (Node.js, React) Namaste Technologies No office location \$45K - \$90K REMOTE javascript node.js
- Senior Software Engineer, Backend

I could have intervened in this function at the source code level, changing the installerPackageName variable to a non-zero value, and then compiling and running it on the Android operating system. But, as Bill Gates said in an interview, "I always hire the laziest person because they find the shortest way to do the job", I decided to take the lazy way out and look for a shortcut. :)

I'm sure most security researchers who come across such a situation prefer to use the Frida toolkit, but I believe that life should not be limited to Frida, so I decided to look for an alternative tool and a different way. After a short search on Google, I remembered Xposed Framework, which I had used before in penetration tests, especially to bypass SSL Pinning, and has more than 1400 plugins.

After installing Xposed Framework on the Android Oreo operating system on GenyMotion, I began to look at its plugins. Among the plugins, I was immediately attracted by XPrivaclyLua plugin. As its name suggests, this plugin helps protect your privacy by feeding applications installed on Android with fake information (such as fake location information). It works by roughly hooking onto functions that try to collect this information and providing fake information instead of real information. To shape the plugin to your needs, you need to buy the Pro version and create scripts using Lua programming language.

Xposed Installer

Xposed Status 🔴



Xposed Framework version 90-beta3 is active.





INSTALL/UPDATE

Version 90-beta3 🔒

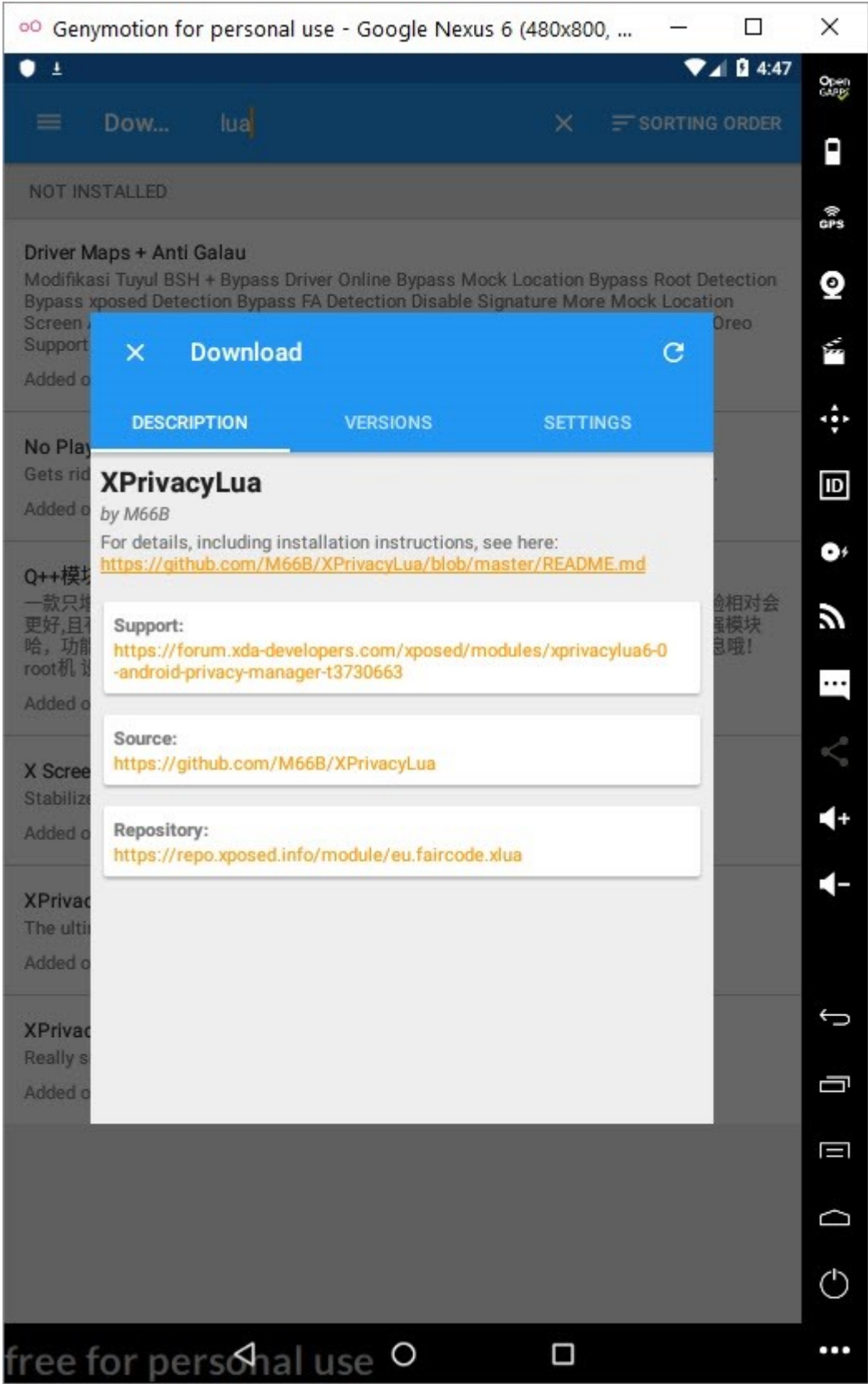
UNINSTALL

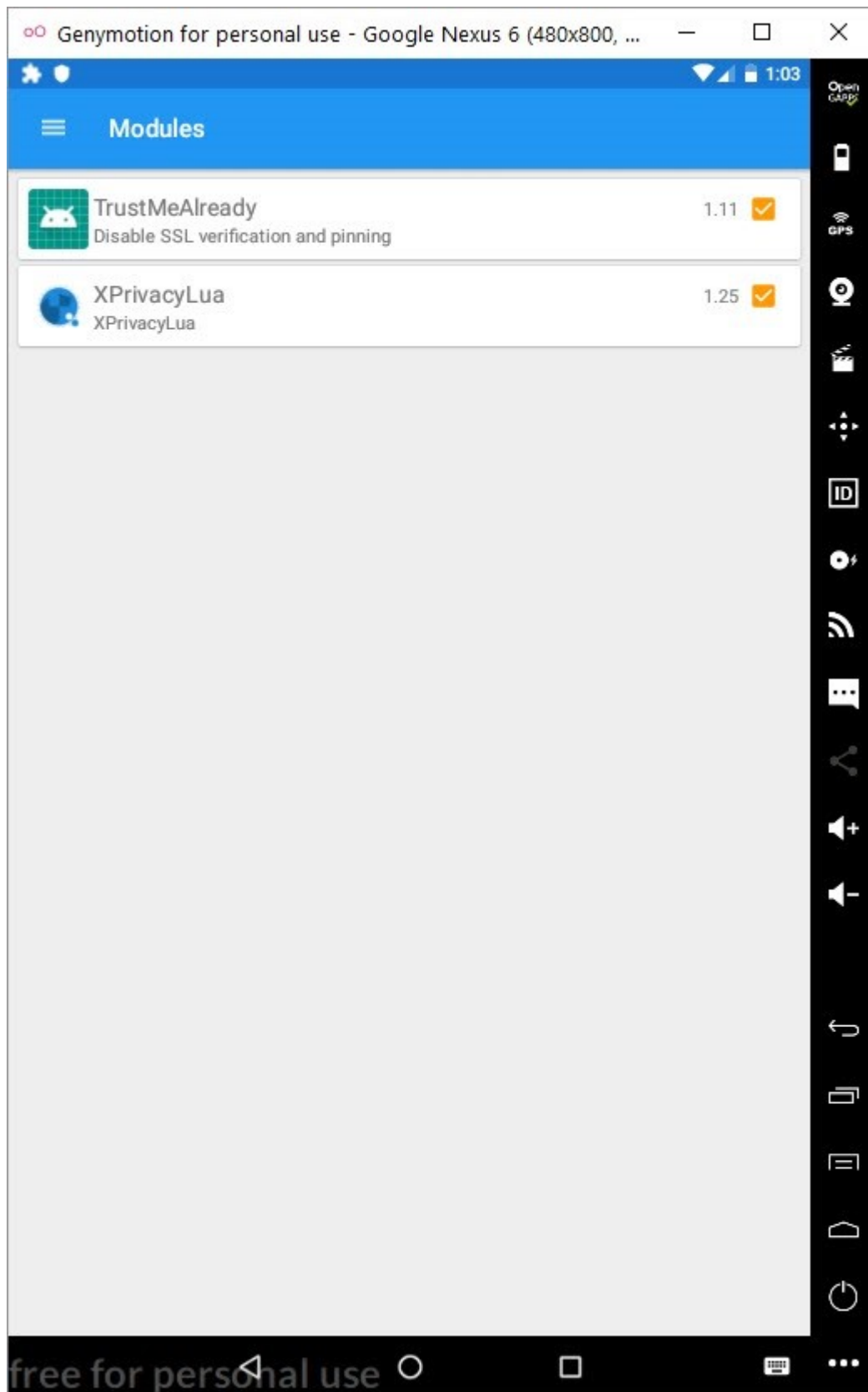
Uninstaller (20180117) ☁️

YOUR DEVICE

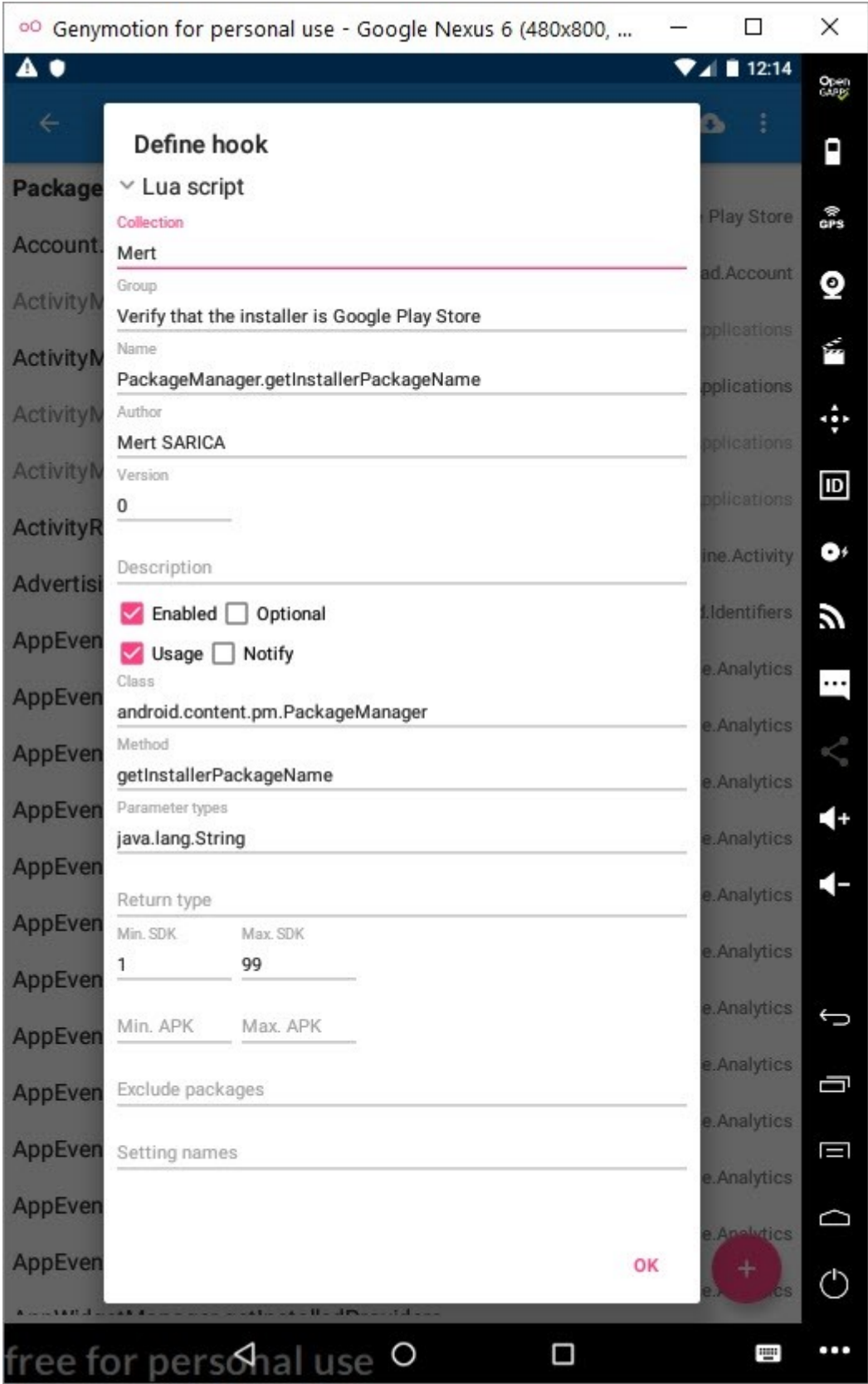
-  Android 8.0.0 (Oreo, API 26)
-  Genymotion Android Google Nexus 6
-  x86
-  Verified Boot is deactivated







After preparing and activating a small script that changes the `installerPackageName` variable using Lua, when I ran the application, I was happy to see that the application no longer displayed the previous warning message and I could view the web traffic using Charles Proxy.



PackageManager.getInstallerPackageName

```

Mert Verify that the installer is Google Play Store

^ Lua script
-- Mert.PackageManager.getInstallerPackageName is a Lua
hook definition
-- designed to work with XPrivacyLua.

-- Mert.PackageManager.getInstallerPackageName is free
software: you can redistribute it and/or modify
-- it under the terms of the GNU General Public License
as published by
-- the Free Software Foundation, either version 3 of the
License, or
-- (at your option) any later version.

-- Mert.PackageManager.getInstallerPackageName is
distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied
warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the
-- GNU General Public License for more details.

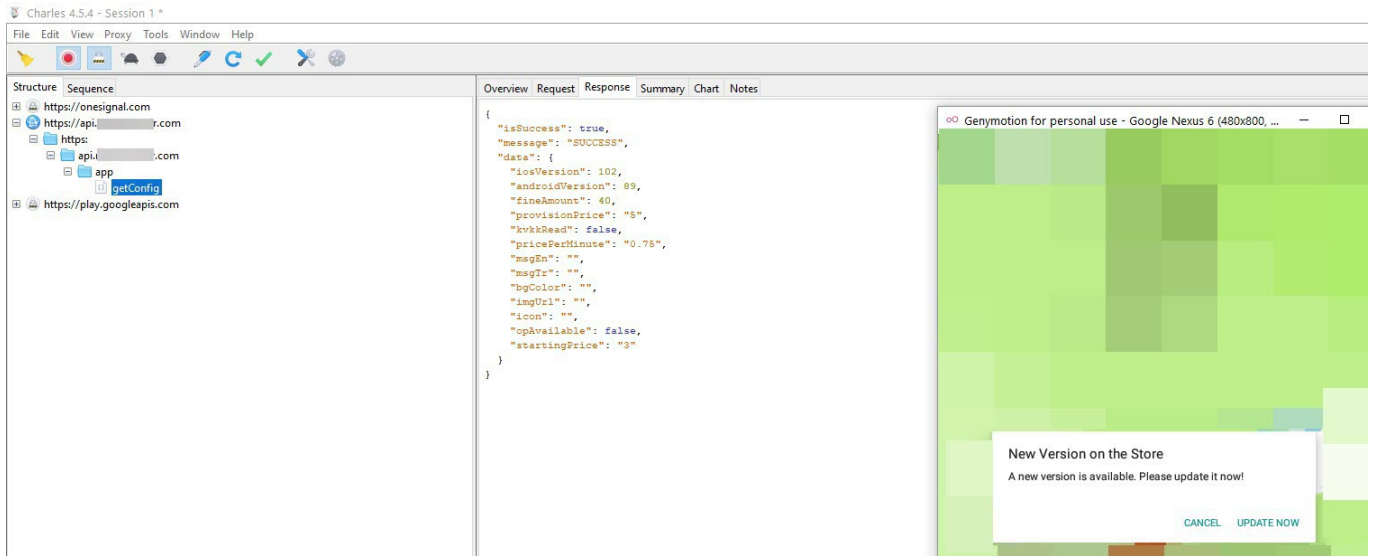
-- You should have received a copy of the GNU General
Public License
-- along with XPrivacyLua. If not, see
<http://www.gnu.org/licenses/>.

-- Copyright (C) 2019 Mert SARICA (www.mertsarica.com)

function after(hook, param)
    local result = param:getResult()
    local fake = 'com.android.vending'
    param:setResult(fake)
    return true, result, fake
end

```

OK



I hope this article will provide an alternative tool and method for those who are looking for alternatives to Frida in security research. Hope to see you in the following articles.