Normal Permissions in Android: An Audiovisual Deception

Deepsec 2017

Constantinos Patsakis and Efthimios Alepis

November 17, 2017

Department of Informatics, University of Piraeus
80, Karaoli & Dimitriou, 18534, Piraeus, Greece.
Prelude
One of the highlights of this talk is the full disclosure of a security issue we reported last February and was patched last month from the Android security team. Sadly, the patch is available only in the Pixel/Nexus security bulletin.
CVE-2017-0807 Detail

Current Description

Source: MITRE  Last Modified: 10/03/2017  View Analysis Description

Impact

CVSS Severity (version 3.0):
- CVSS v3 Base Score: 9.8 Critical
- Impact Score: 5.9
- Exploitability Score: 3.9

CVSS Version 3 Metrics:
- Attack Vector (AV): Network
- Attack Complexity (AC): Low

CVSS Severity (version 2.0):
- CVSS v2 Base Score: 10.0 HIGH
- Vector: (AV:N/AC:L/Au:N/C:C/I:C/A:C) (legend)
- Impact Subscore: 10.0
- Exploitability Subscore: 10.0

CVSS Version 2 Metrics:
- Access Vector: Network exploitable
- Access Complexity: Low
Other bugs

We believe that Android has far more security issues which allow an adversary to steal a lot of sensitive information. Some of these are presented in this talk.

Unfortunately, while we have disclosed some of them more than one year, little or no progress has been made.
As Android users we feel rather uncomfortable knowing that the Security team of Android considers that many of what will be displayed in this talk are “working as intended”, silently patched but not acknowledged, or are “infeasible” etc.

We have reasons to believe that others may have been already exploiting it. Some of these things must be fixed.

For more details: [2, 1].
Acknowledgments

This work was supported by the European Commission under the Horizon 2020 Programme (H2020), as part of the OPERANDO project (Grant Agreement no. 653704) and is based upon work from COST Action CRYPTACUS, supported by COST (European Cooperation in Science and Technology).
Introduction
Google made a radical redesign of its security model in the last versions of Android. Part of them was the introduction of *runtime permissions*.

Users can grant/revoke app *dangerous* permissions.
Dangerous permissions
Normal permissions

These permissions can be regarded as the ones that expose the user or the system to the least possible risk when granted. Therefore, the system automatically grants them at installation, without asking for the user’s explicit approval. They cannot be revoked, and in many cases they cannot be “seen”.
If an app does not use any dangerous permission, then the app is harmless!

It cannot use: camera, mic, phone, storage, location, SMS, calendar or contacts.

How bad can it be?
We assume that a user has been tricked into installing an app. In order not to raise any suspicion and to convince the user to install the app, we do not use any dangerous permission.

The latter app does not ask for any permission, every permission is normal so automatically granted...
Visual illusions
Goal

What you see is **not** what you think.
Android is designed to run in a rather constrained environment in terms of both size and computational resources which imply many constraints for the UI.

Basically, what we have is layers with UI components, which are stacked one on top of the other.
Who is on top?
The layer which is on top is the one that the user sees and interacts, **but** due to size, UI and other OS constraints the user **cannot** determine **which** is the foreground app.

Note that every installed app in Android knows which other apps are installed.
If an adversary knew that you have installed a banking app, he could overlay it and get your credentials. Bankbot, Bankun, Koler, Lockdroid, MazarBot, SlemBunk, and Svpeng exploit such capabilities.
To quantify the problem, according to CheckPoint [5] 74% of ransomware, 57% of adware, and 14% of banker malware abuse the SYSTEM_ALERT_WINDOW permission.
This permission is clearly very dangerous.

According to Google Developer resources [4]: “Very few apps should use this permission; these windows are intended for system-level interaction with the user”.

The permission is poorly implemented leading to well-known attacks such as [9, 6].
I know what you use and when. Usage statistics are very important for many companies. Android has a special permission for monitoring apps through UsageStatsManager.

Android Developer:

This API requires the permission `android.permission PACKAGE_USAGE_STATS`, which is a system-level permission and will not be granted to third-party apps. However, declaring the permission implies intention to use the API and the user of the device can grant permission through the Settings application.
How to make visual illusions

Our tricks:

- Determine foreground app (<Nougat)
- Make the user open an app you control (All Android versions)
- Steal PIN/Pattern. (All Android versions)
- Overlay others apps. (All Android versions before Oreo, Pixel/Nougat branches excluded)

All the above without using any dangerous permission.
Sniffing secure lock PIN/Pattern
Motivation
Where is this information stored?

- The pattern is stored as an unsalted SHA-1 hash in /data/system/gesture.key
- PIN/passphrase are stored in /data/system/password.key as a concatenation of the password’s SHA-1 and MD5 hash values. Contrary to the patterns, the text-based passwords use a salt which is stored in the /data/system/locksettings.db.
How does the user lock his device?

No read/write access to contents of /data/

What is the size of these files?

```
ls -l /data/system/gesture.key
```
Let’s replicate the lock screen! Goal

Use fingerprint or draw unlock pattern

EMERGENCY CALL
Get user’s wallpaper

All applications are allowed to access device’s wallpaper by requesting the `getDrawable` property without the need for declaring any dangerous permission, as we reported in Security Issue 219663.
Since we know how the user unlocks his screen, we know what to show.

We don’t know when to show it, yet...
We create a BroadcastReceiver to listen for screen-off events (ACTION_SCREEN_OFF), while our app is running on the foreground. Our attack is triggered by the user, not when he tries to unlock his device by using the power button, but when he locks it!

Our fake lock screen will be brought to the foreground after the screen-off event and will remain there invisible until the user tries to unlock his smartphone.
The fake lock screen

Use fingerprint or draw unlock pattern

Thank you for giving me your pattern!
Get foreground app
Let’s work the other way around: If I don’t know which is the foreground app, could I find which one is \textbf{not} the foreground app?
Android is Linux powered so it uses procfs to store the data of its processes. Information in this filesystem is well protected, in terms of reading and altering the stored information, but we have side leakages as some metadata are publicly available to *all* applications.
Android runs in mobile devices which have constrained resources, whereas many refinements have been introduced by Google to allow Android to perform resource allocation and release. If there is free memory, Android uses Zygote to launch a new VM through Dalvik or ART. If not let’s *kill* someone!
If there is not any free memory, Android has to close the least used application. Each application is given a oom_adj_score, stored under `/proc/[pid]/`

Pruning all the system applications we can determine which is the application which is less probable to be killed.

Can you “guess” which one is this?
Luring the user
Notifications

From API level 11, one must denote the text of the notification; through `setContentText` which accepts a string variable, the title of the notification; through `setContentTitle` which also accepts a string variable, and the notification icons for the status bar and the notification view, using `setSmallIcon` and `setLargeIcon` respectively.

As of API level 23, both icons can be set dynamically using custom bitmaps. Prior to API level 23, only the `setLargeIcon` provided this feature, as `setSmallIcon` required an integer which denoted the resource ID in the application’s package of the drawable to use.
Fake Notifications

Android Emulator - Nexus_5X_API_255554

Android Emulator - Nexus_5X_API_255554

Android Emulator - Nexus_5X_API_255554

PayPal!
New Special Offer!

PayPal!
New Special Offer!

PayPal!
New Special Offer!

CLEAR ALL

CLEAR ALL

CLEAR ALL
Apps can create home-screen shortcuts using the normal permission INSTALL_SHORTCUT in their manifest. The underlying mechanism to create a shortcut is intents, so the developer has to declare three variables: a string which denotes the caption of the shortcut (EXTRA_SHORTCUT_NAME), a string which denotes the “action” of the intent to be launched (setAction), and its icon as a bitmap (EXTRA_SHORTCUT_ICON).
Overlays
## Who can draw on top of other apps?

<table>
<thead>
<tr>
<th>UI Window Type</th>
<th>Required Permission</th>
<th>Manifest Declaration</th>
<th>Focusable</th>
<th>Duration</th>
<th>Launch from service</th>
<th>Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toast Messages</td>
<td></td>
<td></td>
<td></td>
<td>3.5 sec</td>
<td></td>
<td>[7, 10]</td>
</tr>
<tr>
<td>Alert Messages</td>
<td>SYSTEM_ALERT_WINDOW</td>
<td>✓</td>
<td>No limit</td>
<td></td>
<td></td>
<td>[9, 6]</td>
</tr>
<tr>
<td>System Alerts</td>
<td>SYSTEM_ALERT_WINDOW</td>
<td>✓</td>
<td>No limit</td>
<td></td>
<td>✓</td>
<td>[9, 3, 6]</td>
</tr>
<tr>
<td>Keyguards*</td>
<td>Required</td>
<td>✓</td>
<td>No limit</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Normal activity</td>
<td>Required</td>
<td>✓</td>
<td>No limit</td>
<td></td>
<td>✓</td>
<td>[2]</td>
</tr>
<tr>
<td>Notification</td>
<td></td>
<td>✓</td>
<td>No limit</td>
<td></td>
<td>✓</td>
<td>[8, 2]</td>
</tr>
</tbody>
</table>
Normal activities are in **all** Android apps, versions and flavors. They are the default user interaction elements in Android. Could you resize them? Could they overlay other apps?
Customizing activities

Let’s play with the manifest!

Let’s use the Theme.Translucent.NoTitleBar to remove any possible title. Now let’s make it flow!

```xml
<item name="android:windowIsFloating">true</item>
<item name="android:windowIsTranslucent">true</item>
<item name="android:windowBackground">@android:color/transparent</item>
<item name="android:windowNoTitle">true</item>
```
We override the activity’s onCreate() method. Create WindowManager.LayoutParams object with dimAmount set to 0, flagged with the attributes FLAG_LAYOUT_NO_LIMITS and FLAG_NOT_TOUCH_MODAL.
To position the sized floating activity on the screen, one can fine tune several parameters of the corresponding LayoutParam e.g. “Gravity” parameters, or actual position through (X,Y) on-screen coordinates.
(Semi) Transparent, sizeable activities.

Question: How many of these can I have?
Scenario 1
Does Android detect overlays?

Nope! Only when it comes to app/runtime permissions...
## OS Components Overlay Protection

<table>
<thead>
<tr>
<th>Feature</th>
<th>Protected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Administrators</td>
<td>X</td>
</tr>
<tr>
<td>Optimise Battery Usage</td>
<td>X</td>
</tr>
<tr>
<td>Do not disturb permission</td>
<td>X</td>
</tr>
<tr>
<td>Apps that can appear on top</td>
<td>X</td>
</tr>
<tr>
<td>Vr Helper Services</td>
<td>X</td>
</tr>
<tr>
<td>Change System Settings</td>
<td>X</td>
</tr>
<tr>
<td>Notification Access</td>
<td>X</td>
</tr>
<tr>
<td>Use Premium SMS Services</td>
<td>X</td>
</tr>
<tr>
<td>Allow Unrestricted Data Usage</td>
<td>X</td>
</tr>
<tr>
<td>Usage Data Access</td>
<td>X</td>
</tr>
<tr>
<td>General App Permissions</td>
<td>✓</td>
</tr>
<tr>
<td>Specific App Permissions</td>
<td>✓</td>
</tr>
<tr>
<td>Runtime Permissions</td>
<td>✓</td>
</tr>
</tbody>
</table>
OS Components Overlay Protection (Cont.)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Protected</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION_SETTNGS</td>
<td>X</td>
</tr>
<tr>
<td>ACTION_SECURITY_SETTNGS</td>
<td>X</td>
</tr>
<tr>
<td>ACTION_WIRELESS_SETTNGS</td>
<td>X</td>
</tr>
<tr>
<td>ACTION_WIFI_SETTNGS</td>
<td>X</td>
</tr>
<tr>
<td>ACTION_APN_SETTNGS</td>
<td>X</td>
</tr>
<tr>
<td>ACTION_BLUETOOTH_SETTNGS</td>
<td>X</td>
</tr>
<tr>
<td>ACTION_DATE_SETTNGS</td>
<td>X</td>
</tr>
<tr>
<td>ACTION_LOCALE_SETTNGS</td>
<td>X</td>
</tr>
<tr>
<td>ACTION_INPUT_METHOD_SETTNGS</td>
<td>X</td>
</tr>
<tr>
<td>ACTION_DISPLAY_SETTNGS</td>
<td>X</td>
</tr>
<tr>
<td>ACTION_LOCATION_SOURCE_SETTNGS</td>
<td>X</td>
</tr>
<tr>
<td>ACTION_INTERNAL_STORAGE_SETTNGS</td>
<td>X</td>
</tr>
</tbody>
</table>
How we use it: Partially overlay apps/system activities
All Android apps!

When we reported the issue, all Android versions we tested (Oreo preview as well) were vulnerable. We did not test below 4 :/
OVERLAYS

OVERLAYS EVERYWHERE
Automated exploit lifecycle

1. Malory
2. Google Play
3. User's device
4. Firebase
5. Icons
6. Text notifications
   UI elements
   APKs
   Commands

Android Version
Installed Packages
User Activity/Input
Audio Illusions
Using the mic

To use the mic, one needs to request the corresponding dangerous permission.
Exploiting features

Well, what if I don’t care about your voice, but about *what* you say? Android has a feature to allow developers get voice input from the users and convert it to text. Speech-to-Text. You can use it via intents.
So what?

You create an app, (it can be invisible or in a service like activity) and fire the event for Speech to Text regularly to get what the user says in a text format.

SMEAGOL SHOWS THEM SECRET WAYS THAT NOBODY ELSE COULD FIND, AND THEY SAY SNEAK.
Now that we have a mechanism to get data from the user’s output we can extract additional data from the user.

Guess who else can be fired without permissions? How do you send a command? Text to Speech (and some sleep time)!
Exploit ideas

- “Ok Google, send my location to Alice”
- “Ok Google, what is my next appointment”
- “Ok Google, call 1 2 3 4 5 6 7”
- “Ok Google, read my last SMS”
- Open camera with intent, use Text to Speech to say “Shoot” and snap a photo (you need storage though to get the photo).

- Initiate video calls, post to IM or social media.

The list is endless as everyone rushes to use them with their apps.
Some of the permissions Google Assistant has

(full list has 84 permissions!)

<table>
<thead>
<tr>
<th>READ_CONTACTS</th>
<th>WRITE_SETTINGS</th>
<th>READ_SMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE_CONTACTS</td>
<td>MEDIA_CONTENT_CONTROL</td>
<td>RECORD_AUDIO</td>
</tr>
<tr>
<td>ACCESS_NETWORK_STATE</td>
<td>ACCESS_FINE_LOCATION</td>
<td>SEND_SMS</td>
</tr>
<tr>
<td>CAPTURE_AUDIO_HOTWORD</td>
<td>CALL_PHONE</td>
<td>GET_TASKS</td>
</tr>
<tr>
<td>USE_CREDENTIALS</td>
<td>CALL_PRIVILEGED</td>
<td>REAL_GET_TASKS</td>
</tr>
<tr>
<td>MANAGE_ACCOUNTS</td>
<td>READ_CALL_LOG</td>
<td>WRITECALENDAR</td>
</tr>
<tr>
<td>WRITE_EXTERNAL_STORAGE</td>
<td>CAMERA</td>
<td>INTERNET</td>
</tr>
<tr>
<td>DOWNLOAD_WITHOUT_NOTIFICATION</td>
<td>WRITE_SMS</td>
<td>USE_FINGERPRINT</td>
</tr>
</tbody>
</table>
Special caution

“Ok Google, read my last SMS”

TWO FACTOR

AUTHENTICATION
You can issue commands to other assistants and get their output.
Demo time!
Thank you for your attention
Questions?
kpatsak@unipi.gr
www.cs.unipi.gr/kpatsak
https://androidsp.cs.unipi.gr
E. Alepis and C. Patsakis.  
**Monkey says, monkey does: Security and privacy on voice assistants.**  

Efthimios Alepis and Constantinos Patsakis.  
**Trapped by the ui: The android case.**  
(To appear).
Yair Amit.
Accessibility clickjacking the next evolution in android malware that impacts more than 500 million devices. https://www.skycure.com/blog/accessibility-clickjacking/, 2016.

Android Developer.
Manifest.permission – SYSTEM_ALERT_WINDOW. https://developer.android.com/reference/android/Manifest.permission.html#SYSTEM_ALERT_WINDOW.
Date retrieved: 28/03/2017.
Check Point Mobile Research Team. 
Android permission security flaw.

Yanick Fratantonio, Chenxiong Qian, Simon Chung, and Wenke Lee.
Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop.
Marcus Niemietz and Jörg Schwenk. 
UI redressing attacks on Android devices. 

Zhi Xu and Sencun Zhu. 
Abusing notification services on smartphones for phishing and spamming. 
Lingyun Ying, Yao Cheng, Yemian Lu, Yacong Gu, Purui Su, and Dengguo Feng.  
**Attacks and defence on android free floating windows.**  

Cong Zheng, Wenjun Hu, Xiao Zhang, and Zhi Xu.  
**Android toast overlay attack: “cloak and dagger” with no permissions.**  
2017.