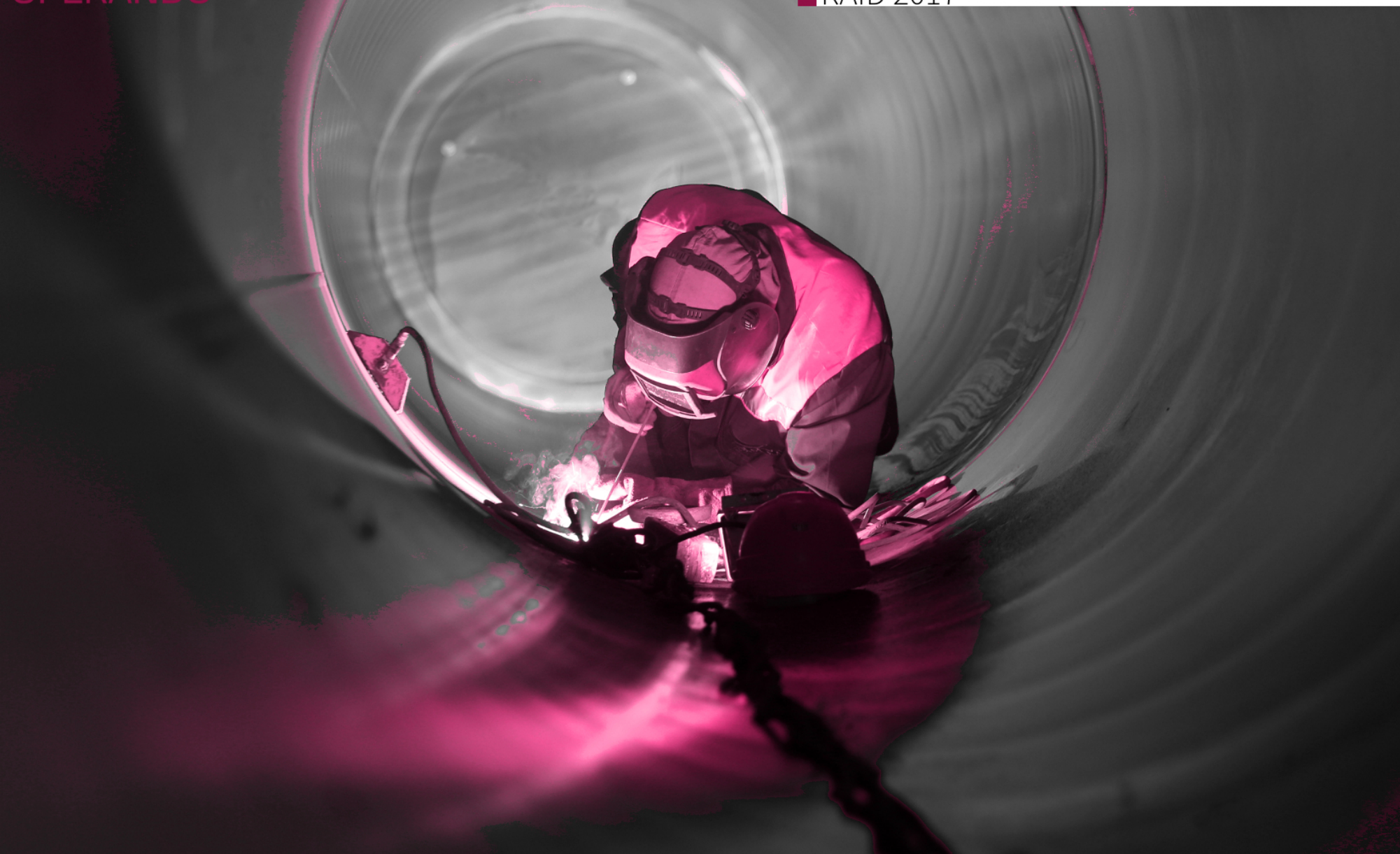
{P}
OPERANDO

Trapped by they UI: The Android case

Efthimios Alepis & Constantinos Patsakis
Department of Informatics
University of Piraeus

RAID 2017



Prelude

Apologies

Apologies to all of **you** attending this talk as we will **not** go to full disclosure. The Android security team is going to provide a patch in the next few days (October 2nd) for CVE 2017-0807.

However, we don't now who is going to receive it as **all** Android versions we tested are vulnerable.

Hope you will like the videos :)

When the update is available we will share everything: details, code etc. The paper is **not** available in the proceedings in your USBs.

Thanks

We would like to thank the organisers of the conference for their support and patience. Special thanks to (in alphabetical order):

- Manos Antonakakis
- Michael Bailey
- Marc Dacier
- Michalis Polychronakis

Introduction

Presumption

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.

VS

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

Android UI

Android is designed to run in a rather constrained environment it 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?

Does it matter?

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.

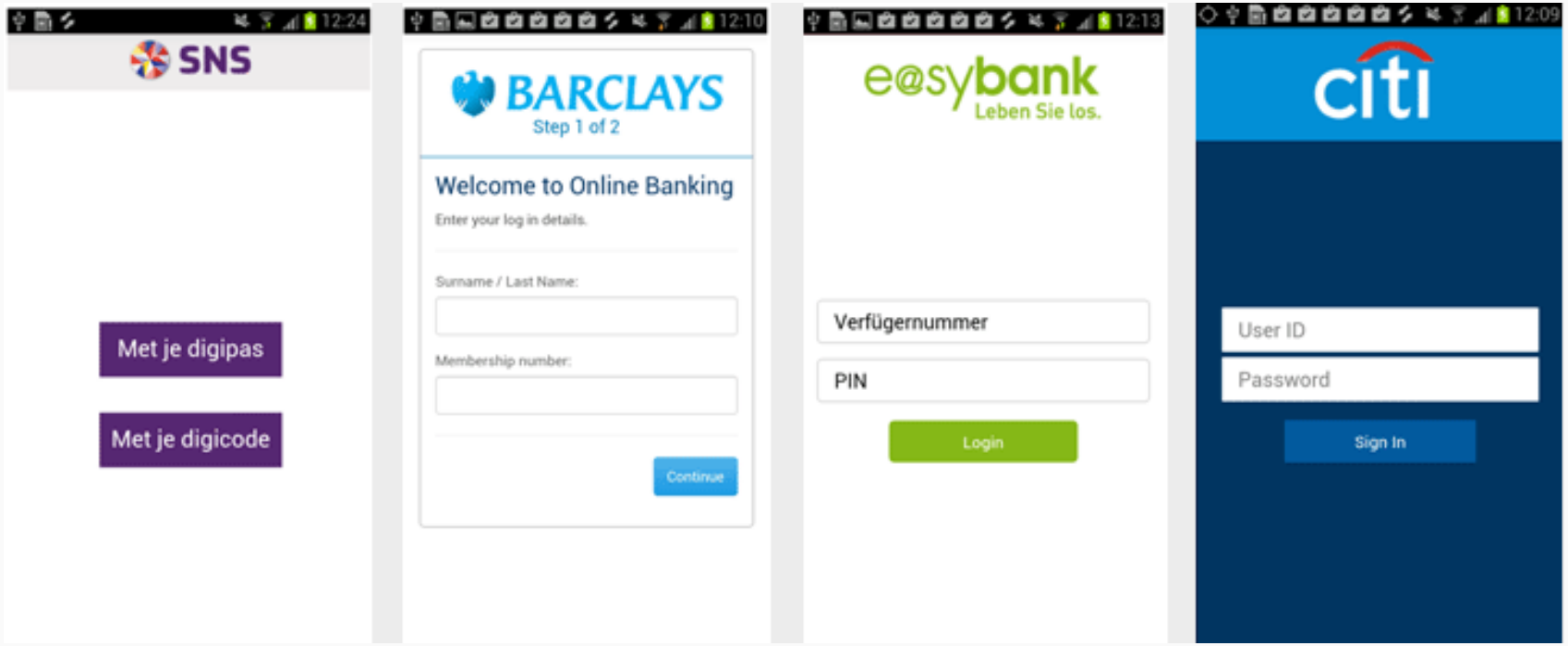
Note that every installed app in Android knows which other apps are installed.

So what?

If an adversary knows 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.

Example



To quantify the problem, according to CheckPoint [4] 74% of ransomware, 57% of adware, and 14% of banker malware abuse the `SYSTEM_ALERT_WINDOW` permission.

The notorious SYSTEM_ALERT_WINDOW

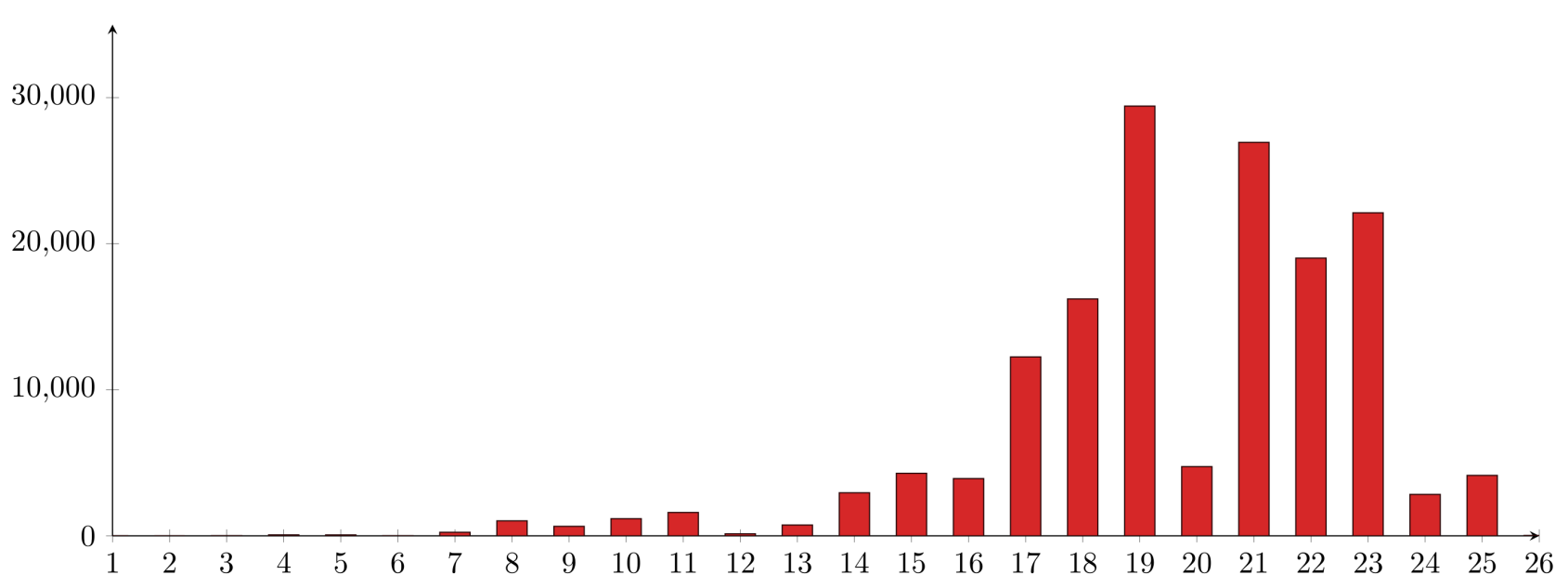
This permission is clearly **very** dangerous.

According to Google Developer resources [3]: "*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 [8, 5].

SYSTEM_ALERT_WINDOW detection

It can be **easily** detected by just looking at the permissions in the manifest so the set of apps exploiting it can be significantly bounded.



Data from Tacyt (`https://www.elevenpaths.com/technology/tacyt/`)

Foreground app

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.*

Foreground app–Our way

You either know it or you lure the user to disclose it.

What's the story in this work?

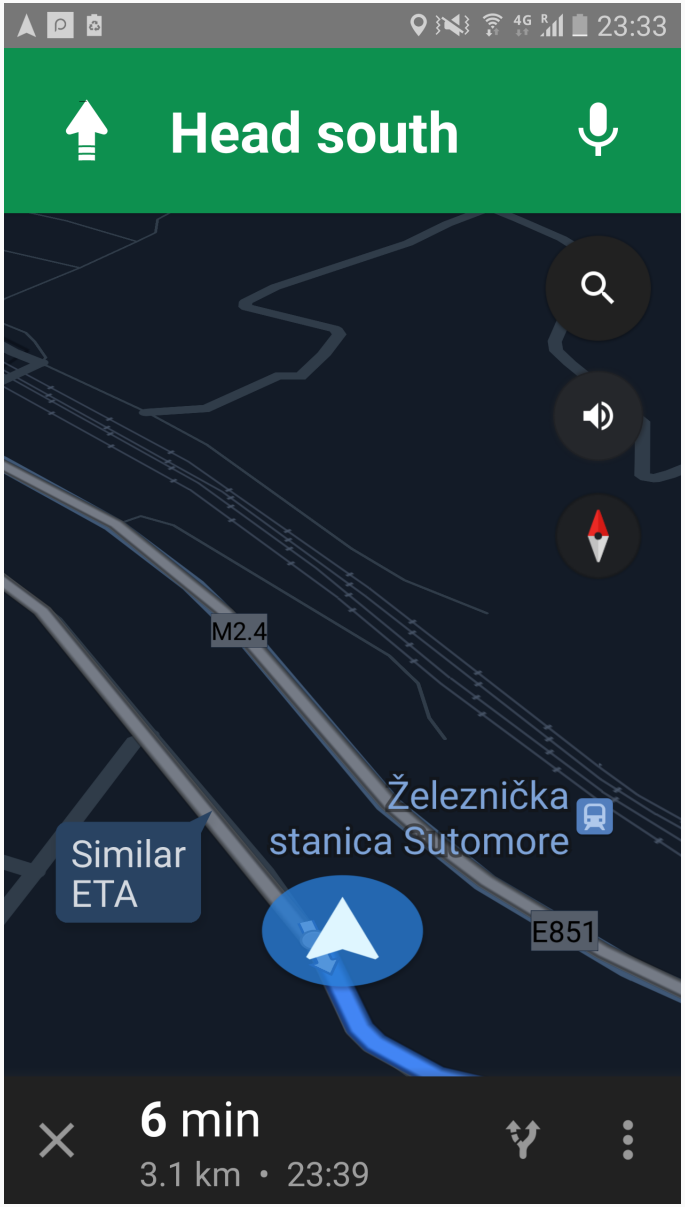
We show:

- How to determine foreground app (<Nougat)
- How to lure the user to disclose the foreground app (All Android versions)
- How to steal PIN/Pattern. (All Android versions)
- How to overlay others apps. (All Android versions)
- A revised version of Tapjacking. (All Android versions)
- Becoming administrator. (All Android versions)

All the above **without** using any dangerous permission.

Sniffing secure lock PIN/Pattern

Motivation


23:33
Head south
M2.4
Železnička
stanica Sutomore
Similar
ETA
E851
6 min
3.1 km • 23:39

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`.

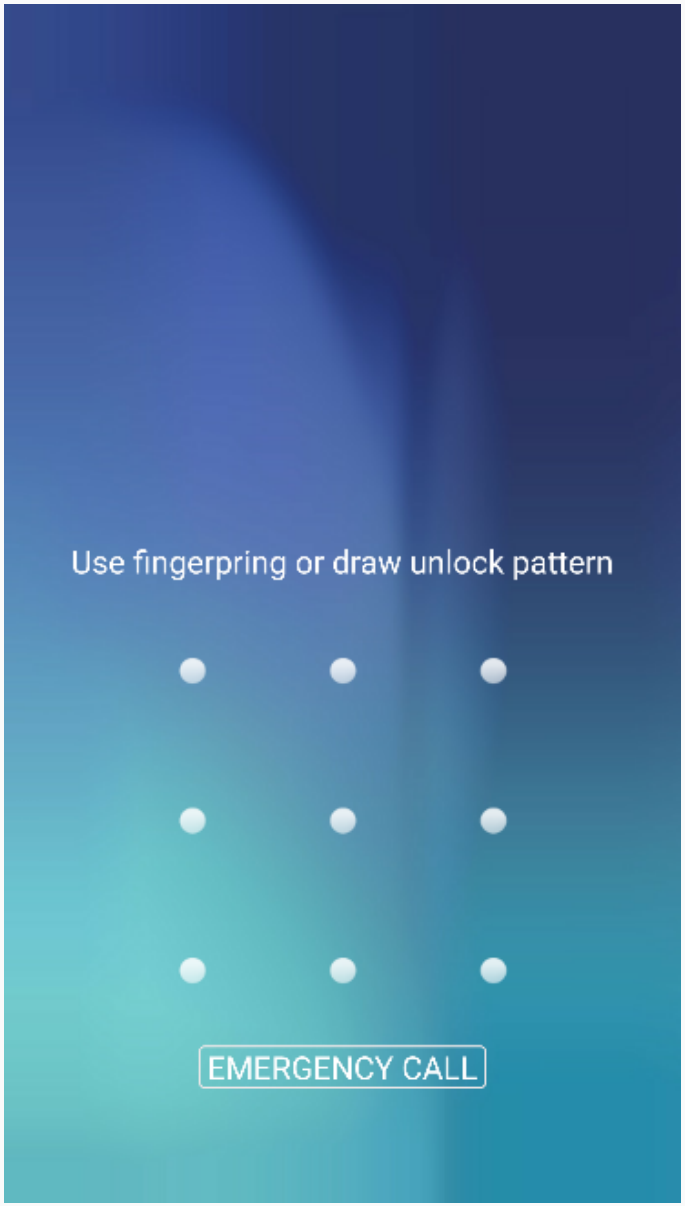
How does the user lock his device?

We don’t have read/write access to contents `/data/` *but* we can “weight” the files!

What is the size of these files?

```
ls -l /data/system/gesture.key
```

Let’s replicate the lock screen! Goal



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, us we reported in Security Issue 219663 (not fixed yet but will be).

Draw the screen

Since we know how the user unlocks his screen we know what to show.

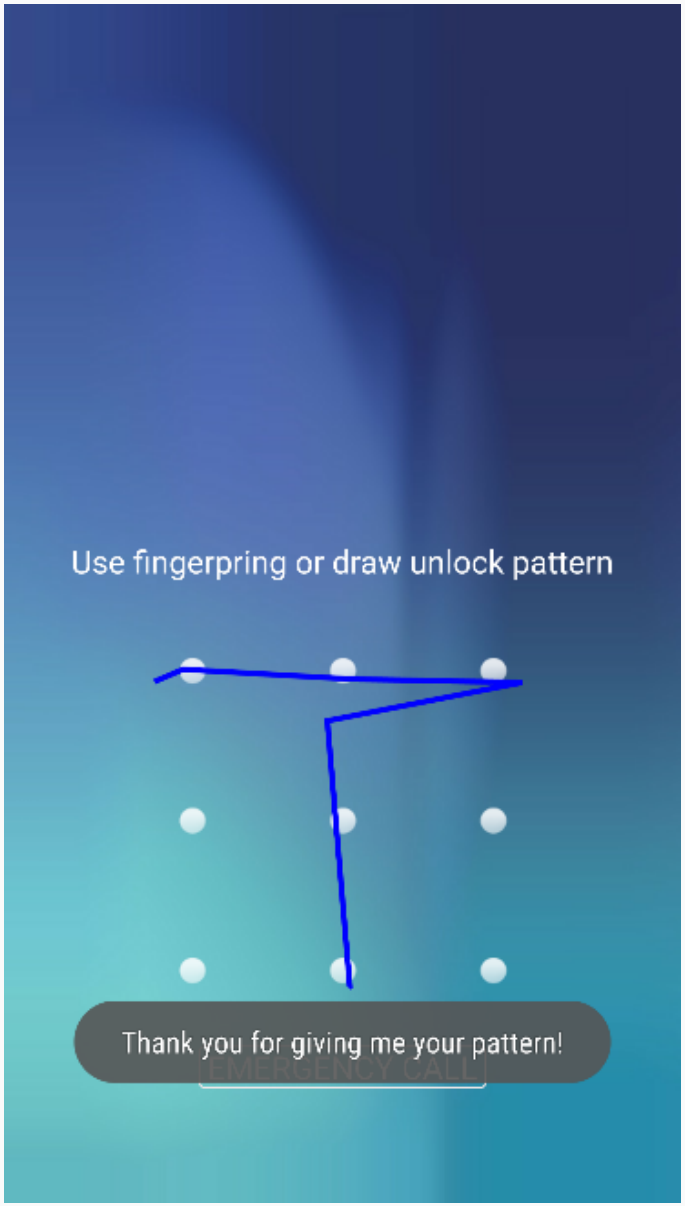
We don’t know when to show it, *yet*...

Presenting the fake lock screen

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!

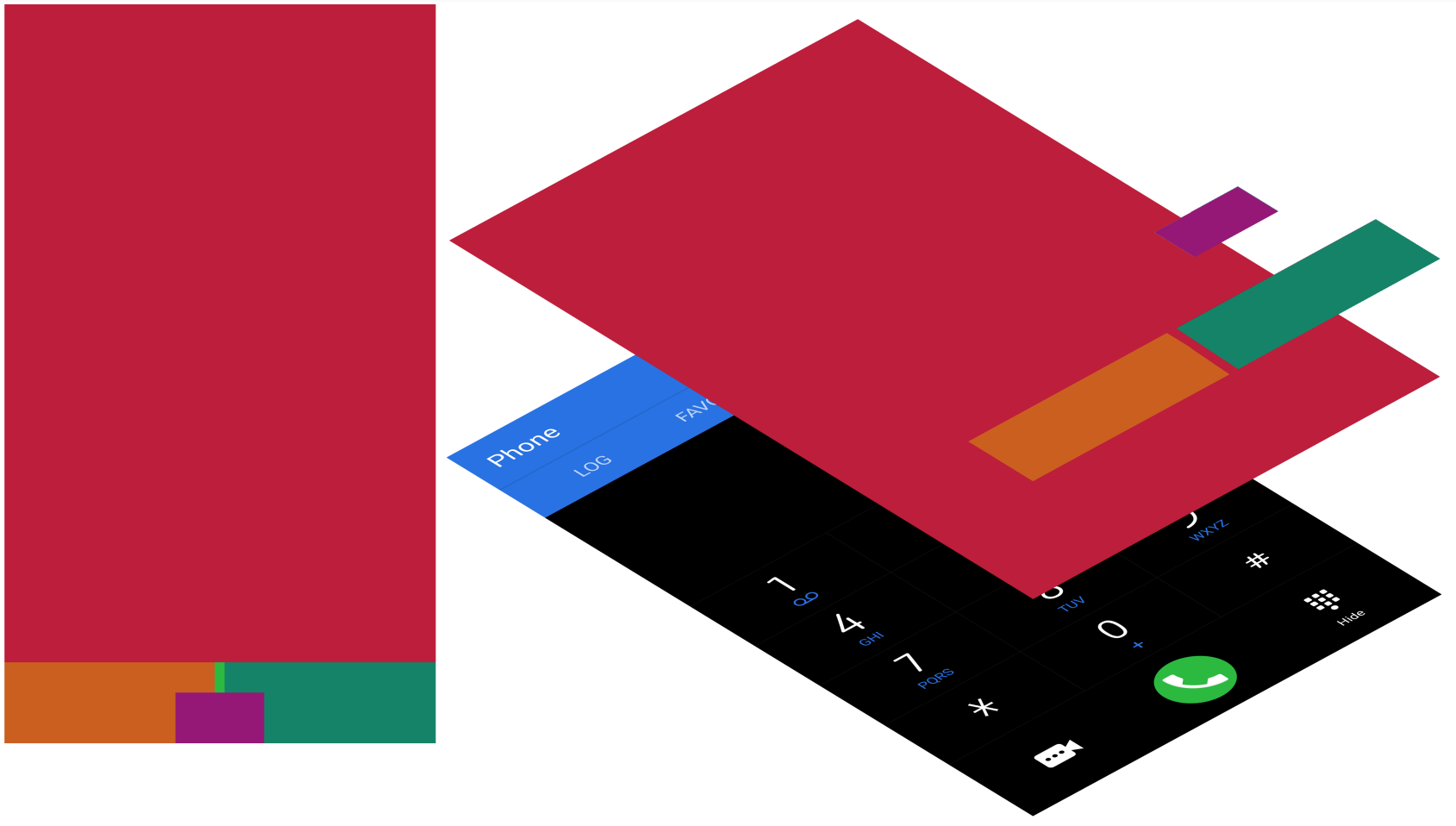
Our fake lock screen will be brought to the foreground after the screen-off event and will remain there invisible until the user tries to unlock his smartphone.

The fake lock screen



Get foreground app

Gameplay



Get foreground activity–Not applicable to Nougat

Let’s work the other way around: If I don’t know which is the foreground app, could I find which one is **not** the foreground app?

Get foreground activity–Not applicable to Nougat

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.

Starting an app

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!

The oom_adj_score file

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?

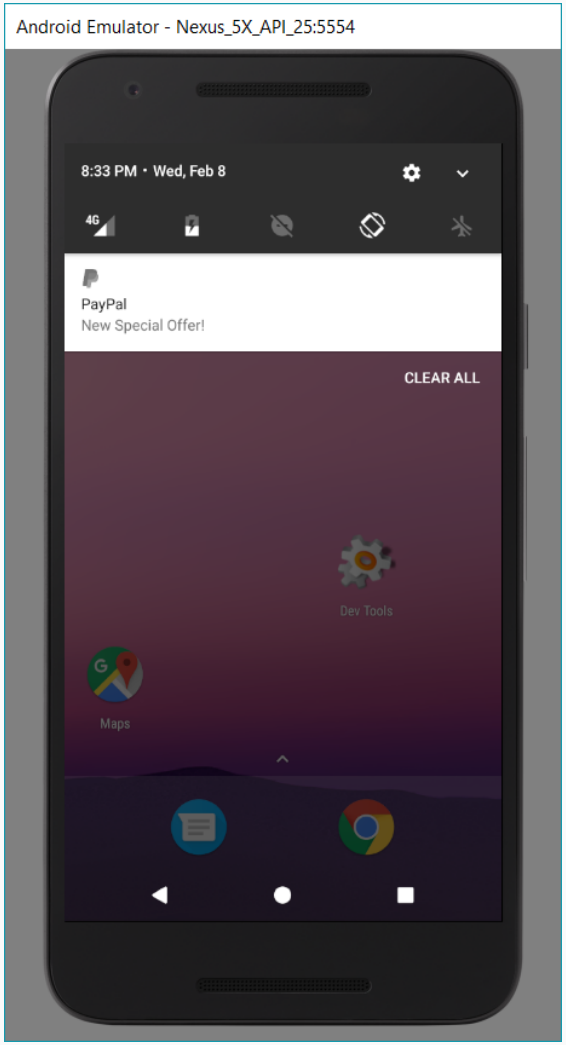
Luring the user

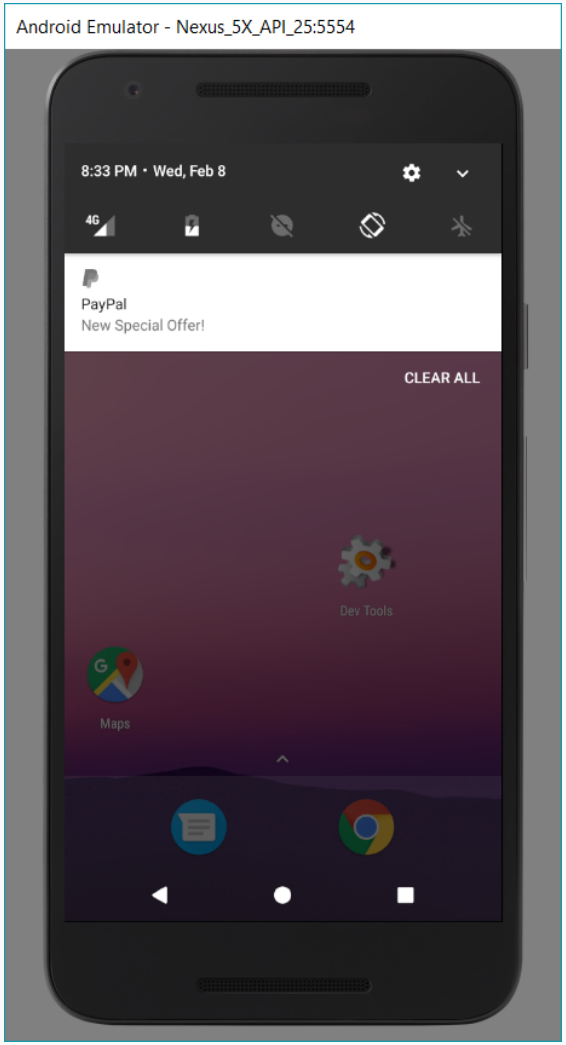
Notifications

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





Shortcuts

Apps can create 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?

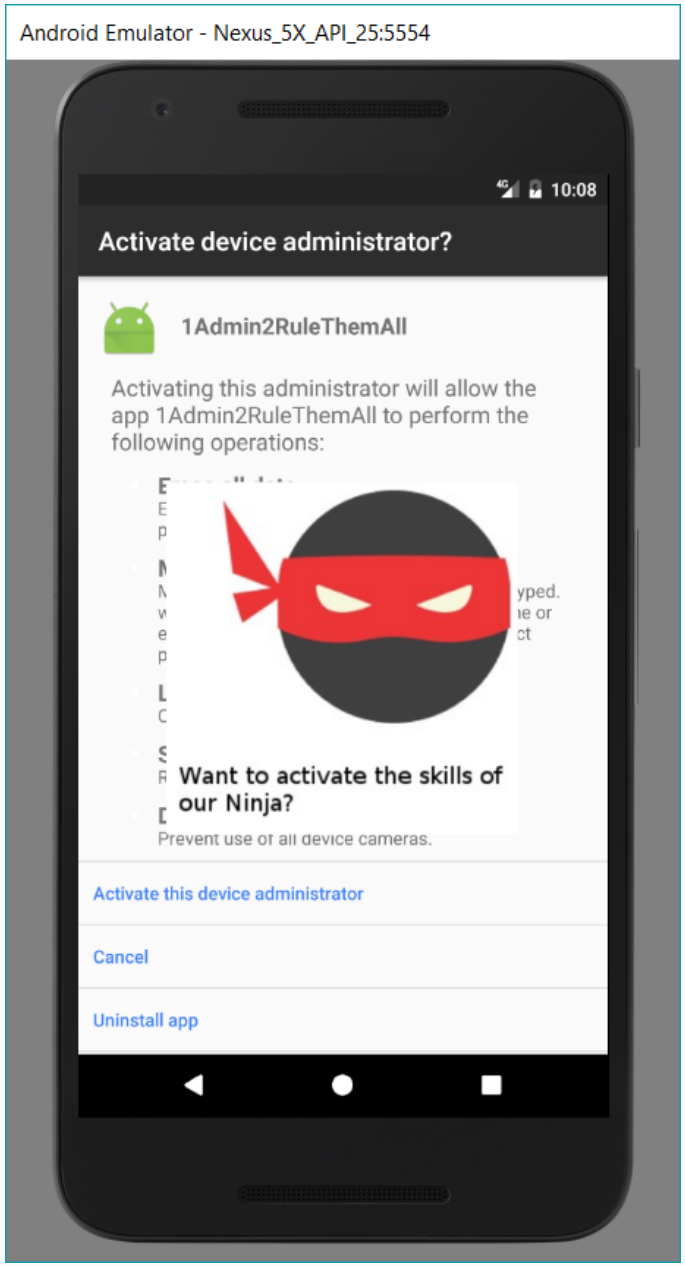
UI Window Type	Required Permission	Manifest Declaration	Focusable	Duration	Launch from service	Attack
Toast Messages				3.5 sec		[6, 9]
Alert Messages	`SYSTEM_ALERT_WINDOW`		✓	No limit		[8, 5]
System Alerts	`SYSTEM_ALERT_WINDOW`		✓	No limit	✓	[8, 2, 5]
Keyguards*		Required	✓	No limit	✓	
Normal activity		Required	✓	No limit	✓	
Notification				No limit	✓	[7, 1]

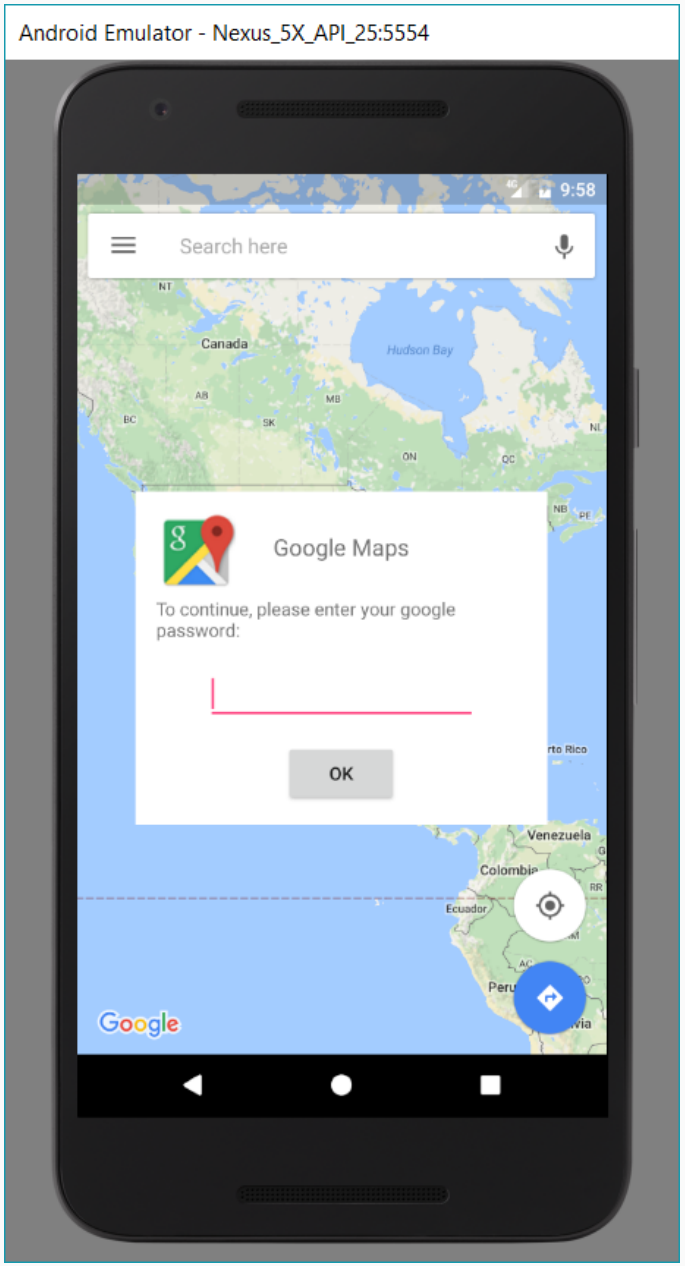
Does Android detect overlays?

Nope! Only when it comes to runtime permissions...

Result

How we use it: Partially overlay apps/system activities





Demo time!

Acknowledgements

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). The authors would like to thank *ElevenPaths* for their valuable feedback and granting them access to Tacyt.

Bloopers

Simpsons did it!

Updates:

Status: Duplicate

Mergedinto: 233504

Comment #2 on issue 234399 by `mrj...@google.com`: Tapjacking revisited
`https://code.google.com/p/android/issues/detail?id=234399`

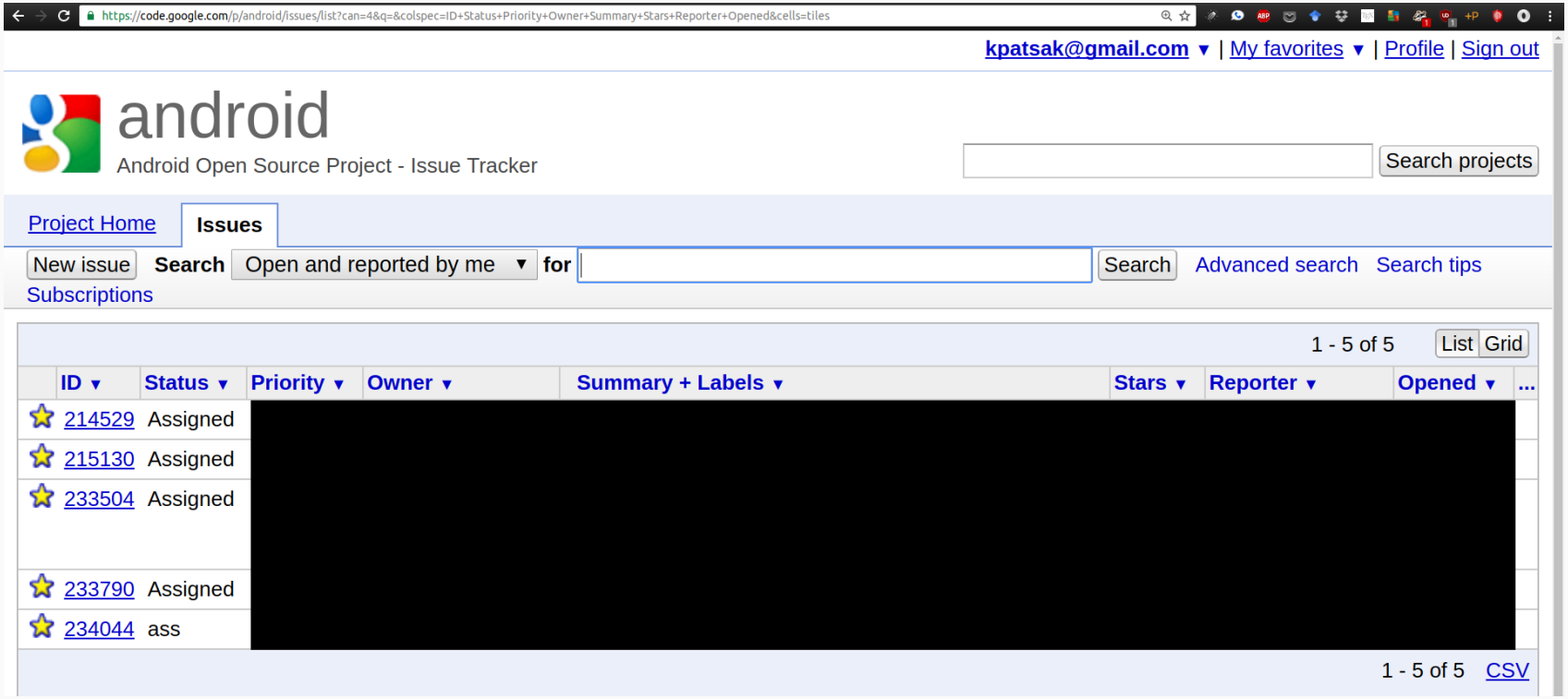
Thank you for reporting this issue.

This issue was already reported by another researcher. The duplicate AOSP id is 233504 and that issue is being tracked by AndroidID-35056974.

Thanks,

WTF! Who managed to do it before us? Ah, it was Tyler Durden!

Pissing people off



Thank you for your attention

Questions?

Efthimios Alepis: talepis@unipi.gr

Constantinos Patsakis: kpatsak@unipi.gr

https://androidsp.cs.unipi.gr

References i

Efthimios Alepis and Constantinos Patsakis.

Trapped by the ui: The android case.

In *Proceedings of the 20th International Symposium on Research in Attacks, Intrusions and Defenses*. Springer, 2017.

(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.

References ii

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.
https://blog.checkpoint.com/2017/05/09/android-permission-security-flaw/, 2017.
Last accessed 9/9/2017.

References iii

Yanick Fratantonio, Chenxiong Qian, Simon Chung, and Wenke Lee.
Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop.
In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, San Jose, CA, May 2017.

Marcus Niemietz and Jörg Schwenk.
UI redressing attacks on Android devices.
BlackHat Abu Dhabi, 2012.

References iv

Zhi Xu and Sencun Zhu.
Abusing notification services on smartphones for phishing and spamming.
In *Proceedings of the 6th USENIX conference on Offensive Technologies*, pages 1–1. USENIX Association, 2012.

Lingyun Ying, Yao Cheng, Yemian Lu, Yacong Gu, Purui Su, and Dengguo Feng.
Attacks and defence on android free floating windows.
In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 759–770. ACM, 2016.

References v

Cong Zheng, Wenjun Hu, Xiao Zhang, and Zhi Xu.
Android toast overlay attack: “cloak and dagger” with no permissions.
https://researchcenter.paloaltonetworks.com/2017/09/unit42-android-toast-overlay-attack-cloak-and-da
2017.
Last accessed 11/9/2017.