Cloak & Dagger
~~From Two Permissions~~ to Complete Control of the UI Feedback Loop

Yanick Fratantonio
joint work with
Chenxiong Qian, Simon Chung, Wenke Lee

Black Hat USA 2017
July 27th, 2017

Who am I?

- PhD candidate @ UC Santa Barbara
  - Graduating in a week! 🎓
  - Soon Assistant Professor at Eurecom, France!

- Research focus on mobile security
  - Program analysis, rowhammer attacks (Drammer), ultrasound cross-device tracking, UI attacks

- As every respectable security researcher, 95% of my time is spent on twitter (@reyammer)

What is this work about?

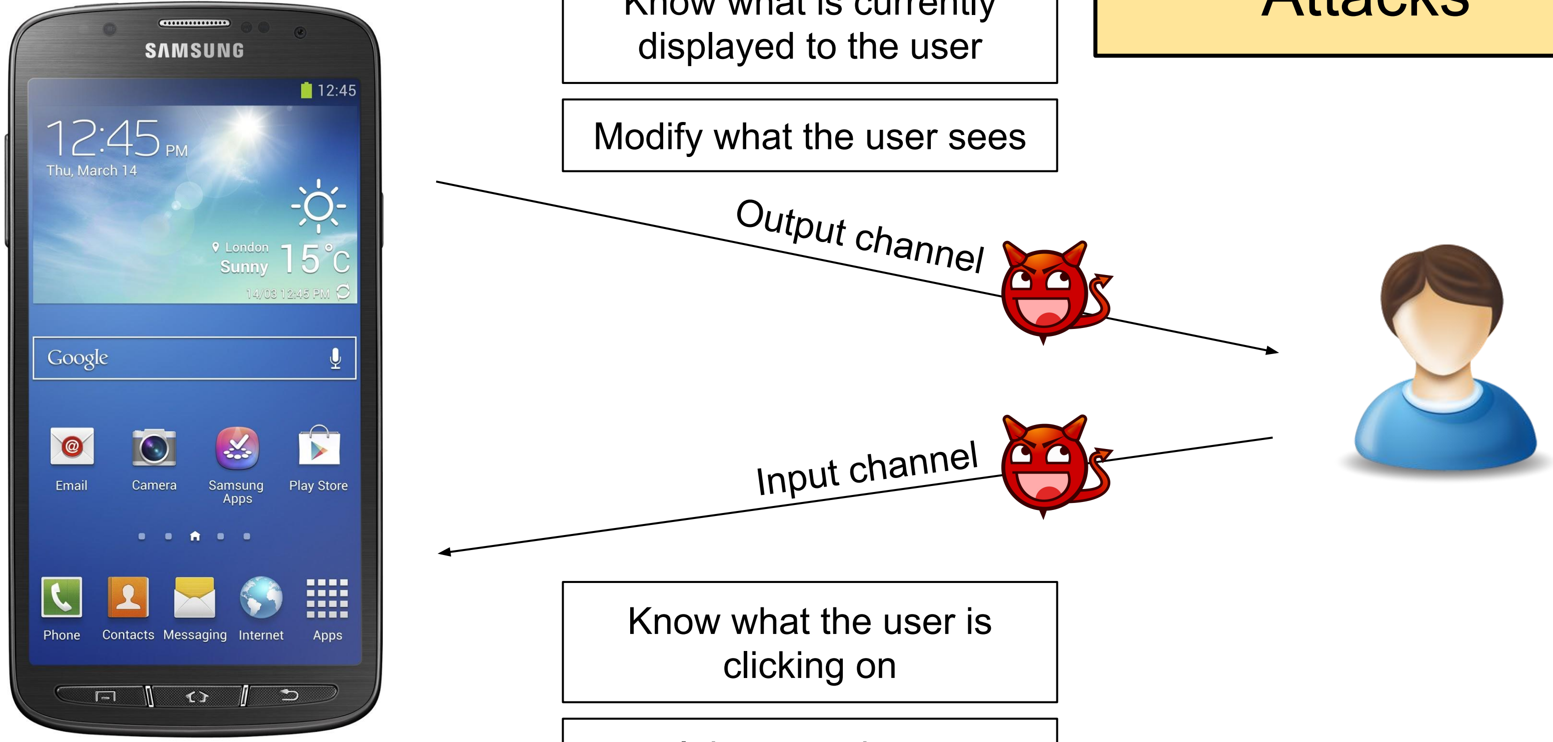
- Cloak & Dagger UI attacks
- Complete control over the UI feedback loop
- Extremely powerful and stealthy…

UI Feedback Loop

Cloak & Dagger Attacks

Know what is currently displayed to the user

Modify what the user sees



Know what the user is clicking on

Inject user input

Why should I care about UI bugs? ACADEMIC BS!

- Android features tons of low-level security mechanisms
  - Sandboxing & permissions
  - Exploit mitigation techniques
  - Attack surface reduction

Good stuff!

BH USA 2017 talk
"Honey, I shrunk the attack surface – Adventures in Android security hardening"
(by Nick Kralevich)

- Some UI bugs can bypass all low-level mechanisms
  - If you can click like a user...confused deputy!
  - "Dear Settings app, I hope this request finds you well.
    Would you mind granting me all permissions? Thx <3"

Two Permissions

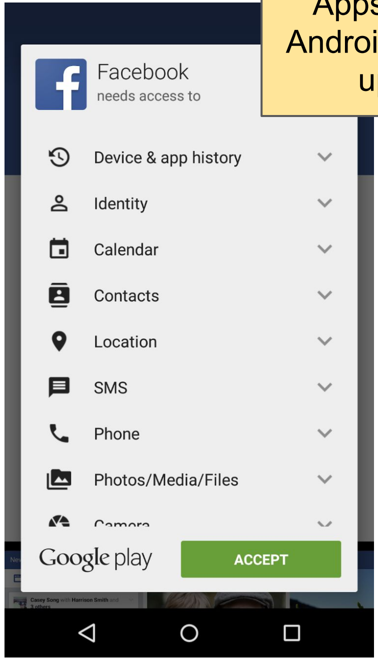
Android Permission System 101

- Android apps are run in a sandbox
- Security capabilities are granted through “permissions”
- Permissions are declared in the app’s manifest (XML file)

```
<manifest>
    <uses-permission
        android:name="android.permission.CALL_PHONE" />
    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
        android:name="android.permission.INTERNET" />
    <uses-permission
        android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
```

Development Phase

Granting Permissions

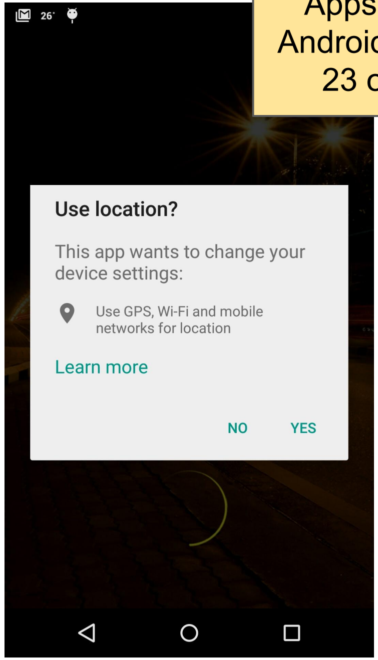


Facebook
needs access to

Device & app history
Identity
Calendar
Contacts
Location
SMS
Phone
Photos/Media/Files
Camera

Apps targeting Android SDK level up to 22

Installation Time Granting



Use location?

This app wants to change your device settings:

Use GPS, Wi-Fi and mobile networks for location

Apps targeting Android SDK level 23 or higher

Runtime Granting

SYSTEM_ALERT_WINDOW (“draw on top”)

- Draw arbitrary windows/overlays on top of the screen
    - Can be completely custom: shape, content, transparency
    - Can be clickable **xor** passthrough

- This permission is used quite often
    - 454 out of 4,455 top apps (10.2%)

- Used by Facebook, Skype, Uber, LastPass, ...

BIND_ACCESSIBILITY_SERVICE (a11y)

- Mechanism for apps to assist users with disabilities
- Many powerful capabilities
  - It is notified for each UI event
  - It can inject UI events (e.g., clicks)
- Several security mechanisms to avoid abuse
- Used by 24 top apps out of 4,455
  - Password managers (LastPass), antivirus apps, app lockers, ...

These two permissions are enough to compromise your device

Why would a user grant these permissions?

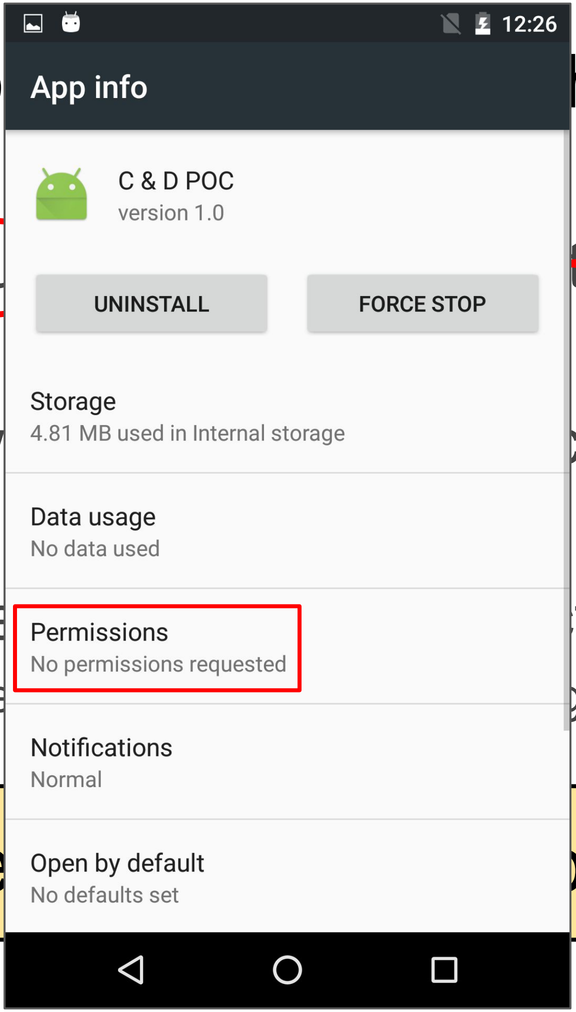
- ~~“The user needs to explicitly approve! Not stealthy!”~~
- “Draw on top” is automatically granted for Play Store apps!
- We developed a new practical clickjacking attack
    - The user is lured to unknowingly enable the a11y!

The list of permissions is not even shown!

Why wo   hes

- ~~“The u~~   ~~tly a~~   ~~y!”~~
- “Draw   cally   ore apps!
- We de   tica
  - The   gly e

The   ons   wn!


12:26

App info

C & D POC
version 1.0

UNINSTALL
FORCE STOP

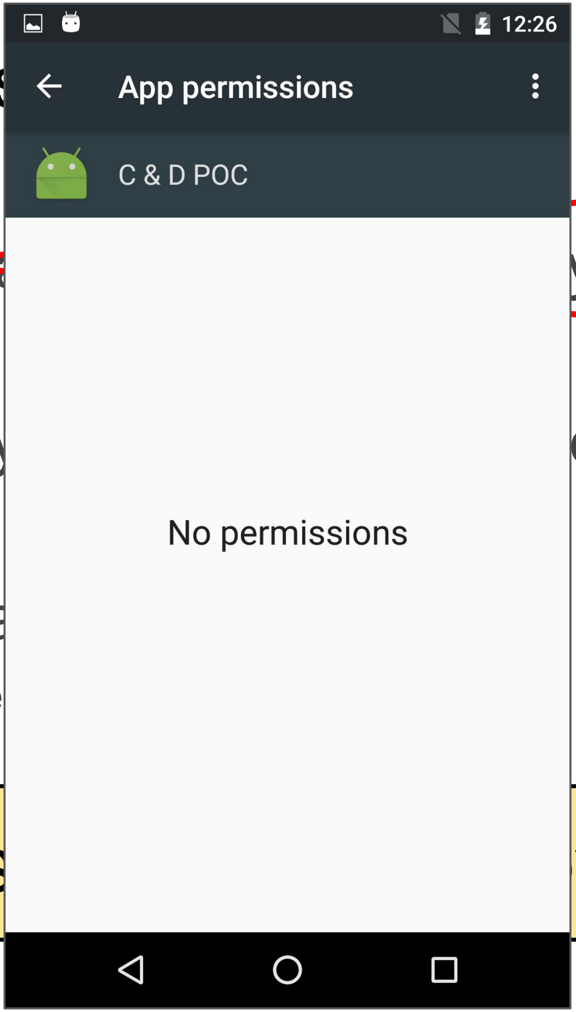
Storage
4.81 MB used in Internal storage

Data usage
No data used

Permissions
No permissions requested

Notifications
Normal

Open by default
No defaults set


12:26

App permissions

C & D POC

No permissions

Security Mechanisms

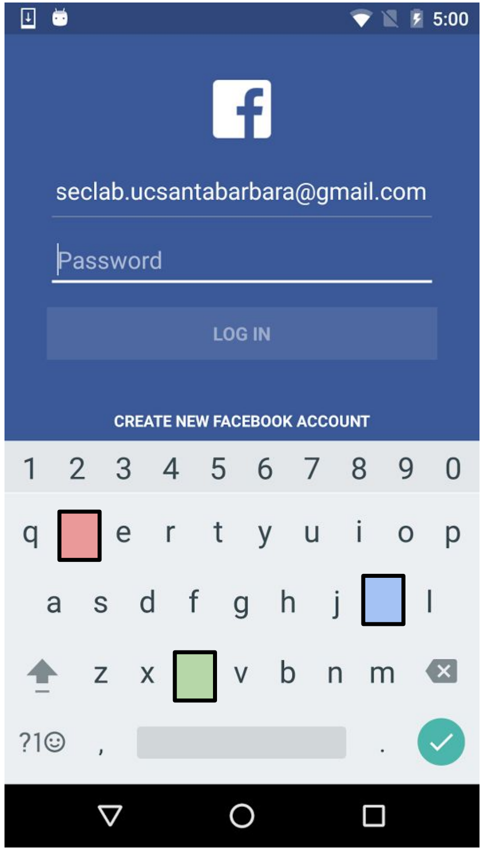
Security Mechanism #1

- For each click on an overlay, only ***one*** of these holds:
    1) The click is “captured” by the overlay
        - The overlay knows when/where the user clicked
    2) The click goes “through” the overlay
        - The click reaches what’s “behind” it
        - The overlay does ***not*** know when/where the user clicked

- No “capture & propagate” click

- Why?

Security Mechanism #1



Security Mechanism #1

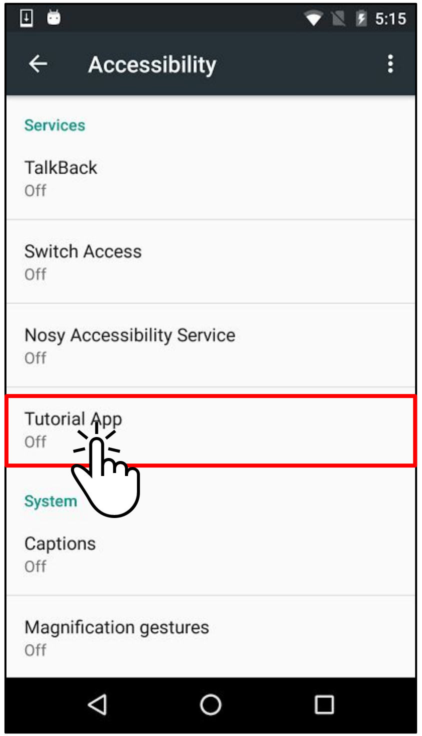
- One possible attack: FLAG_WATCH_OUTSIDE_TOUCH
    - An overlay can receive events even for clicks that land ***outside*** itself

- The click coordinates are set to (0,0) if the click does not reach the app that created the overlay

Security Mechanism #2

- Several steps are required to enable accessibility service

Security Mechanism #2



5:15

Accessibility

Services

TalkBack
Off

Switch Access
Off

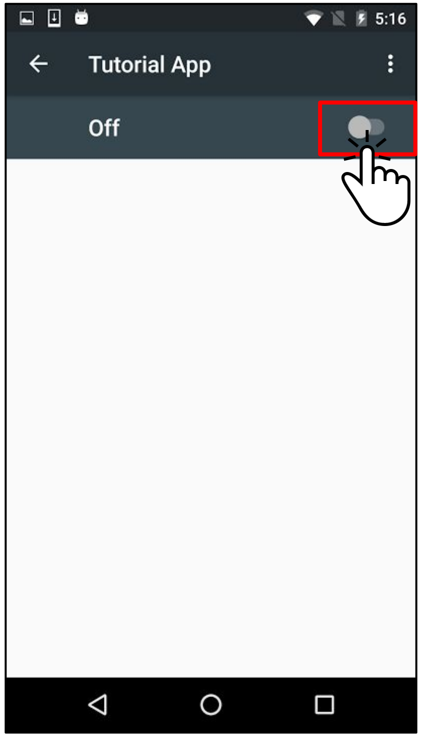
Nosy Accessibility Service
Off

Tutorial App
Off

System

Captions
Off

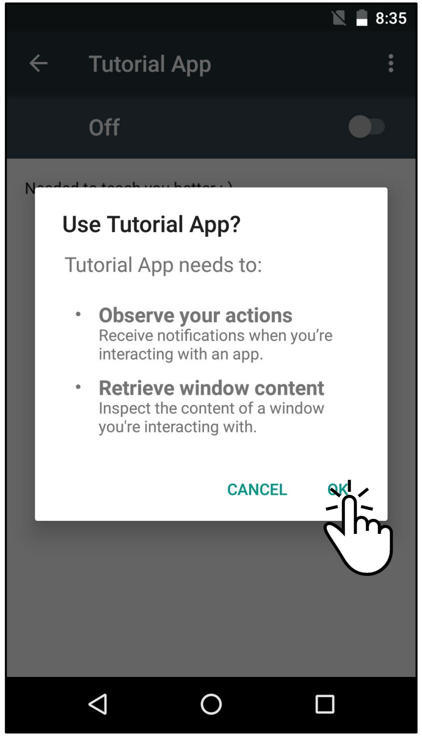
Magnification gestures
Off



5:16

Tutorial App

Off



8:35

Tutorial App

Off

Needed to teach you better :)

Use Tutorial App?

Tutorial App needs to:

- **Observe your actions**
  Receive notifications when you're interacting with an app.
- **Retrieve window content**
  Inspect the content of a window you're interacting with.

CANCEL OK

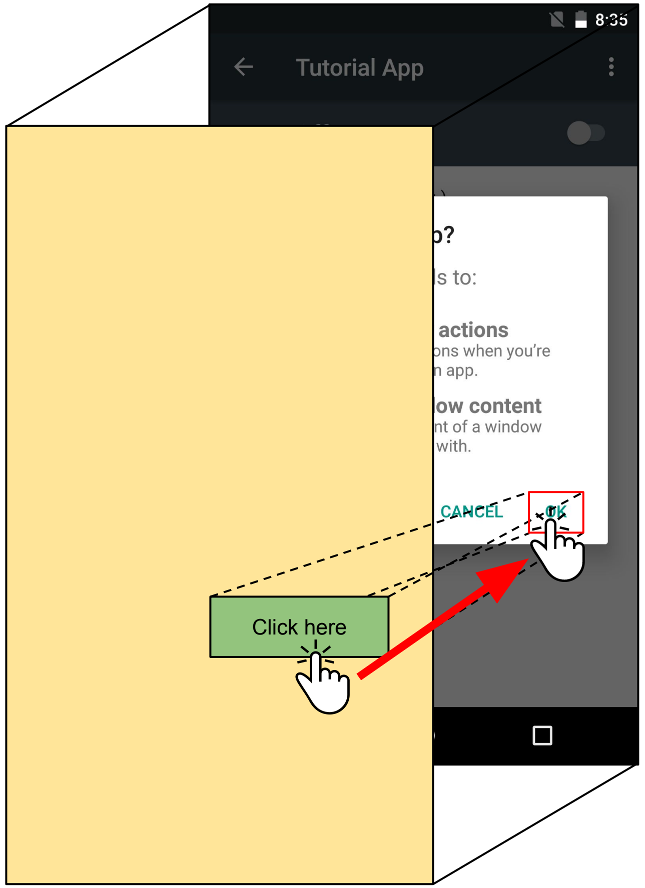
Security Mechanism #3

- Protection against clickjacking

- Google introduced the “obscured” flag
  - When the user clicks on a widget, FLAG_WINDOW_IS_OBSCURED is set if “an overlay was covering the receiving widget”
  - An app can decide to “not trust” the click

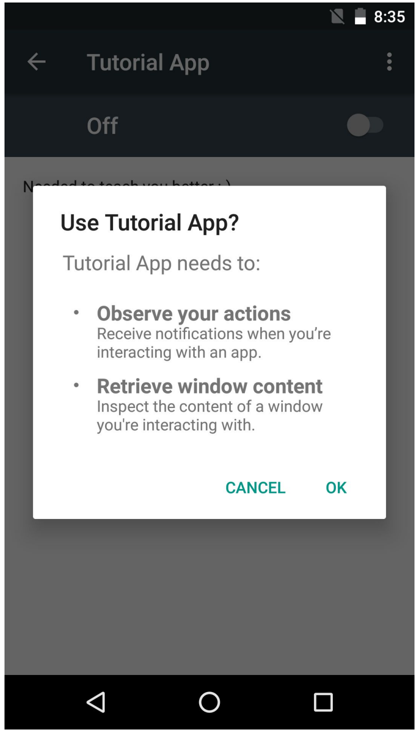
- Another option: `setFilterTouchesWhenObscured()`

Clickjacking 101



UI Redressing Attacks on Android Devices Revisited
Niemietz & Schwenk
BH ASIA 2014

Security Mechanism #3



8:35

Tutorial App

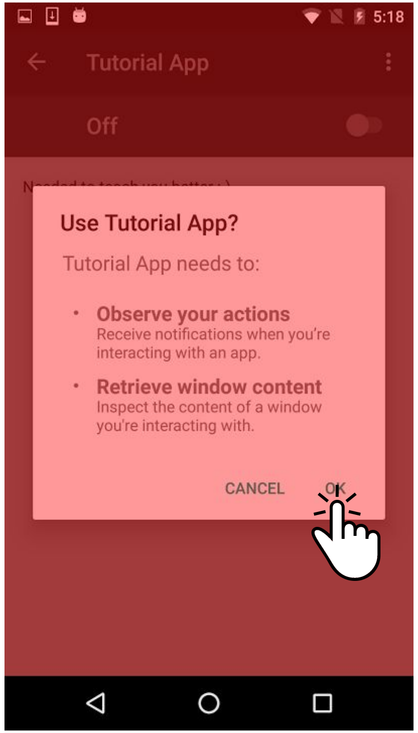
Off

Use Tutorial App?

Tutorial App needs to:

- **Observe your actions**
  Receive notifications when you're interacting with an app.
- **Retrieve window content**
  Inspect the content of a window you're interacting with.

CANCEL OK



5:18

Tutorial App

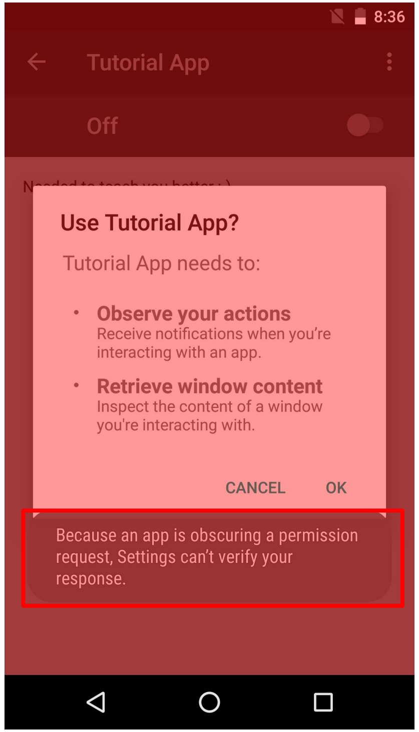
Off

Use Tutorial App?

Tutorial App needs to:

- **Observe your actions**
  Receive notifications when you're interacting with an app.
- **Retrieve window content**
  Inspect the content of a window you're interacting with.

CANCEL OK



8:36

Tutorial App

Off

Use Tutorial App?

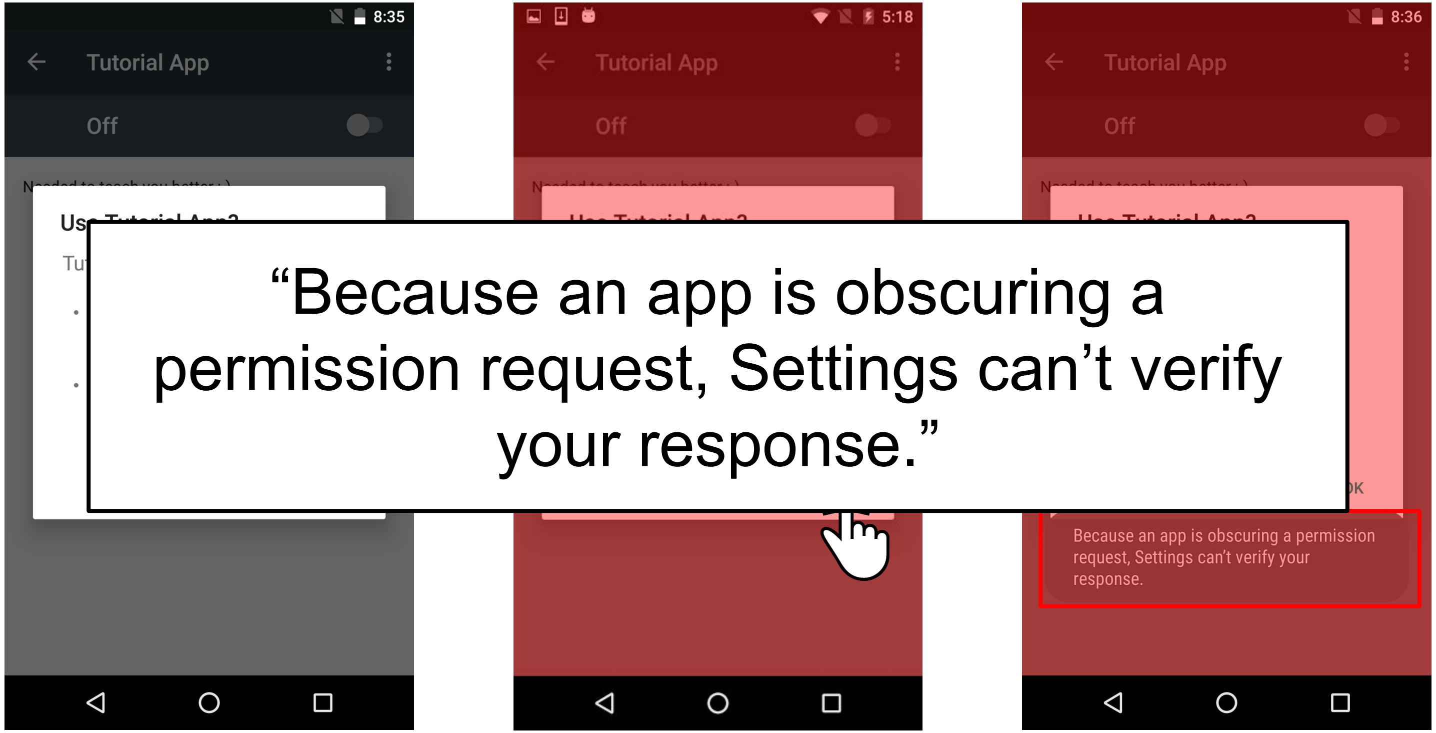
Tutorial App needs to:

- **Observe your actions**
  Receive notifications when you're interacting with an app.
- **Retrieve window content**
  Inspect the content of a window you're interacting with.

CANCEL OK

Because an app is obscuring a permission request, Settings can't verify your response.

Security Mechanism #3



Because an app is obscuring a permission request, Settings can't verify your response.

"Because an app is obscuring a permission request, Settings can't verify your response."

Security Mechanism #4

- Accessibility service ***cannot*** read “sensitive information” off the screen.
- Example: password fields

“Since an event contains the text of its source ***privacy can be compromised by leaking sensitive information*** *such as passwords. To address this issue* ***any event fired in response to manipulation of a PASSWORD field does NOT CONTAIN the text of the password.****”*

Security Mechanisms - Summary

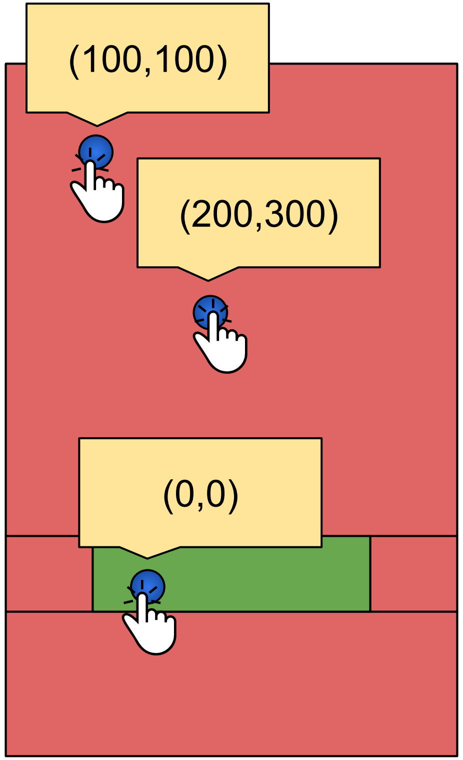
- No “capture & forward”
- Multi-step a11y enabling process
- No clickjacking
- No password stealing

Unleashing Mayhem

Attack: Context-aware Clickjacking

- Multi-stage clickjacking are challenging
  - When to transition to the next stage?
  - What if the user clicks “somewhere else”?
- Security mechanisms
  - The malicious app is not notified about the clicks
  - If the FLAG_WATCH_OUTSIDE_TOUCH is used, the click’s coordinates are set to (0,0) if click lands on another app: where did the user clicked?
- What if there is only “one way” for a click to not reach the malicious app?

Attack: Context-aware Clickjacking



🟥 Clicks do **NOT** go through

🟩 Clicks go through

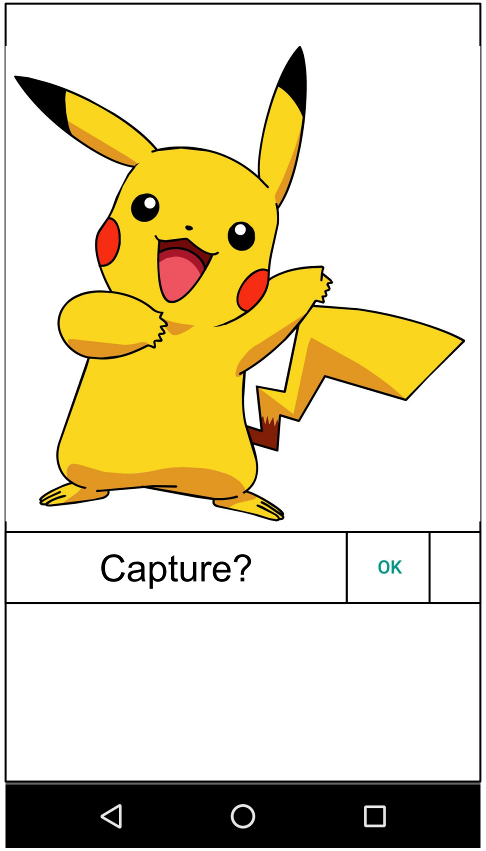
Security mechanism used as side-channel!

Obscured Flag Bypass

- The “obscured” flag is helpful to detect that “another overlay is on top”

- Who says that you need to cover the “target” widget?

Obscured Flag Bypass



Context-Hiding Attack

Attack: Context Hiding

- Design shortcoming: Apps do not have access to enough context information to take informed decisions
- The “obscured flag” is conceptually broken
- Interesting trade-off
  - If the full context is exposed, an attacker could (ab)use this information as side channel to mount phishing attacks

Context-aware clickjacking + Context hiding

- These two attacks are enough to lure the user to enable the accessibility service!

- We just need to hijacking three clicks
    - No guessing is involved
    - The clicks do not need to be consecutive

Back to the “obscured flag”...

- Not only it is not useful…
- ...but it can be abused to mount even worse attacks!

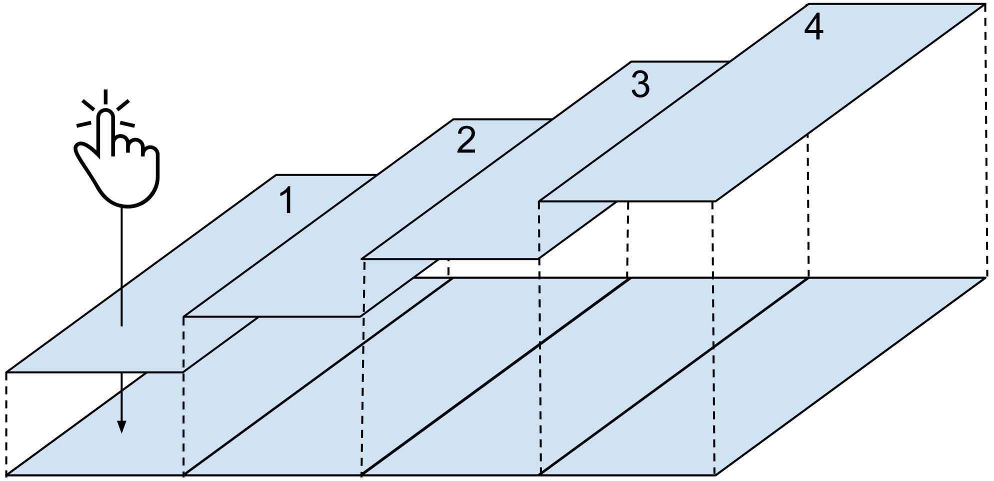
Attack: Invisible Grid Attack

- This attack can record all “keystrokes”
  - It only relies on the “draw on top” permission

- It abuses the “obscured flag” security mechanism

Attack: Invisible Grid Attack

Where did the user click?



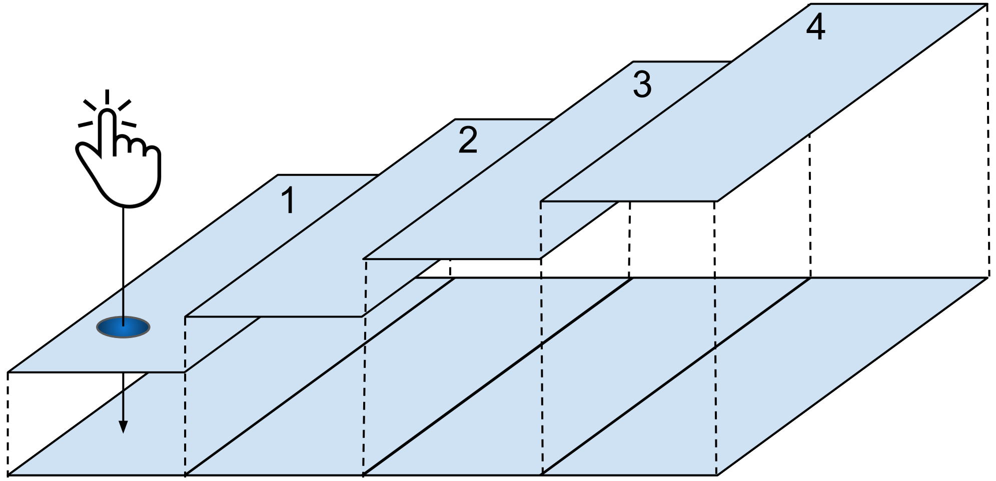
Overlays are drawn
- Invisible
- Clicks passthrough
- FLAG_WATCH_OUTSIDE_TOUCH

Overlay #

1	MotionEvent
2	MotionEvent
3	MotionEvent
4	MotionEvent

Attack: Invisible Grid Attack

Where did the user click?



Overlays are drawn

- Invisible
- Clicks passthrough
- FLAG_WATCH_OUTSIDE_TOUCH

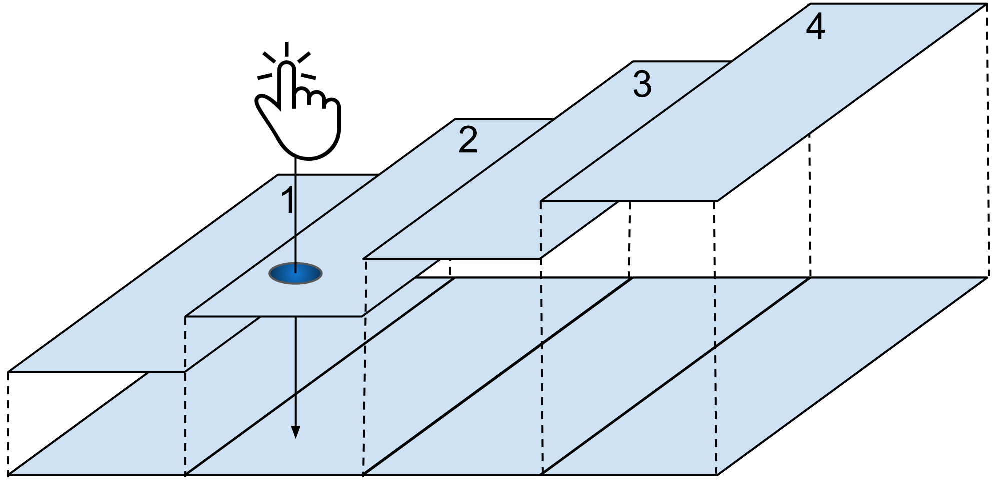
The “obscured” flag is set accordingly!

Overlay #

1	MotionEvent	Not obscured
2	MotionEvent	Not obscured
3	MotionEvent	Not obscured
4	MotionEvent	Not obscured

Attack: Invisible Grid Attack

Where did the user click?



Overlays are drawn
- Invisible
- Clicks passthrough
- FLAG_WATCH_OUTSIDE_TOUCH

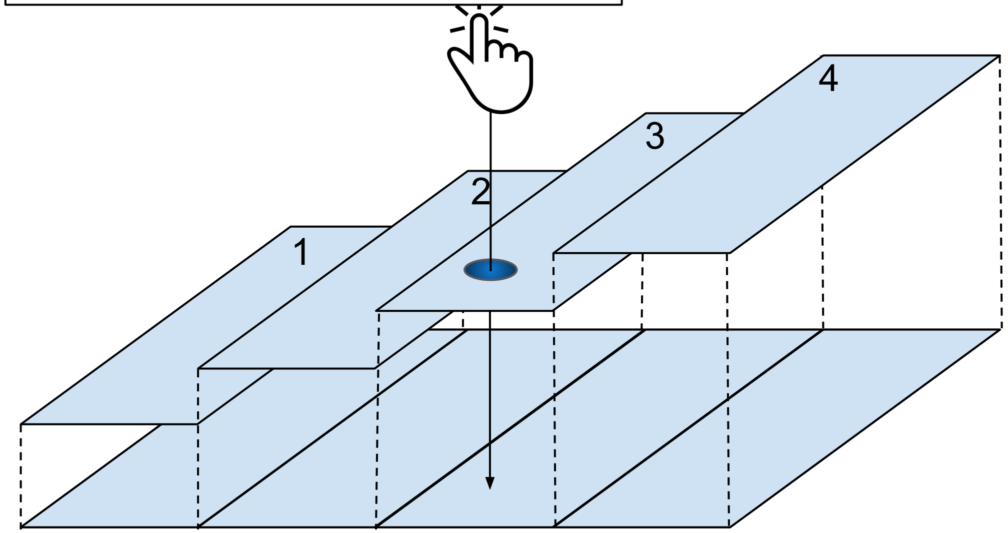
The “obscured” flag is set accordingly!

Overlay #

1	MotionEvent	Obscured
2	MotionEvent	Not obscured
3	MotionEvent	Not obscured
4	MotionEvent	Not obscured

Attack: Invisible Grid Attack

Where did the user click?



Overlays are drawn

- Invisible
- Clicks passthrough
- FLAG_WATCH_OUTSIDE_TOUCH

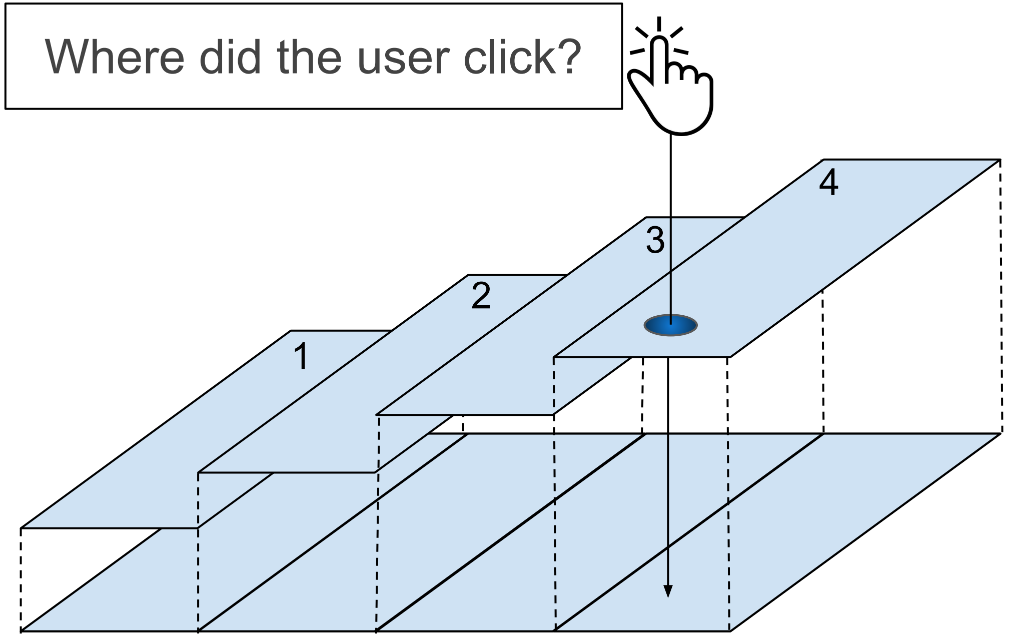
The "obscured" flag is set accordingly!

Overlay #

1	MotionEvent	Obscured
2	MotionEvent	Obscured
3	MotionEvent	Not obscured
4	MotionEvent	Not obscured

Attack: Invisible Grid Attack

Where did the user click?



Overlays are drawn

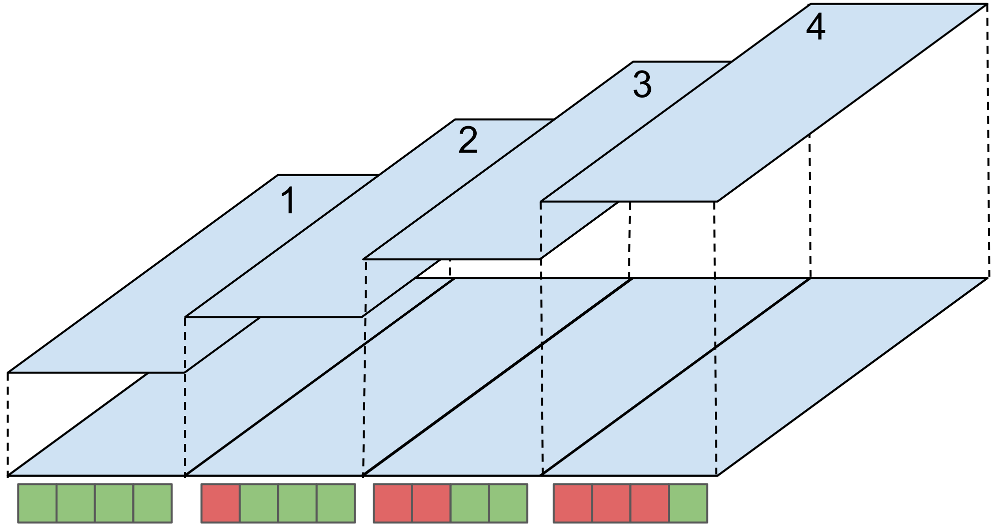
- Invisible
- Clicks passthrough
- FLAG_WATCH_OUTSIDE_TOUCH

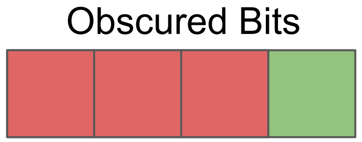
The “obscured” flag is set accordingly!

Overlay #

1	MotionEvent	Obscured
2	MotionEvent	Obscured
3	MotionEvent	Obscured
4	MotionEvent	Not obscured

Attack: Invisible Grid Attack

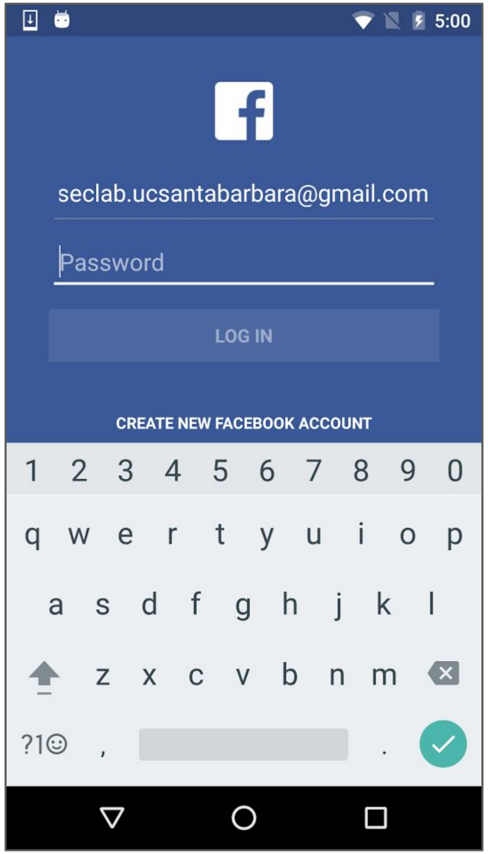




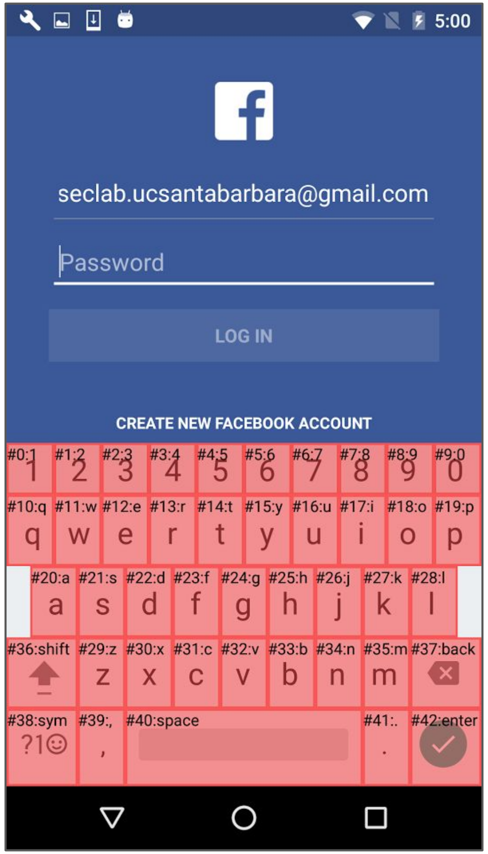
The attacker can use these patterns to infer where the user clicked!

Security mechanism used as side-channel!

Attack: Invisible Grid Attack



Attack: Invisible Grid Attack


5:00
seclab.ucsantabarbara@gmail.com
Password
LOG IN
CREATE NEW FACEBOOK ACCOUNT
#0:1 #1:2 #2:3 #3:4 #4:5 #5:6 #6:7 #7:8 #8:9 #9:0
#10:q #11:w #12:e #13:r #14:t #15:y #16:u #17:i #18:o #19:p
#20:a #21:s #22:d #23:f #24:g #25:h #26:j #27:k #28:l
#36:shift #29:z #30:x #31:c #32:v #33:b #34:n #35:m #37:back
#38:sym #39:, #40:space #41:. #42:enter

These overlays are drawn invisible during a real attack

Design Shortcomings

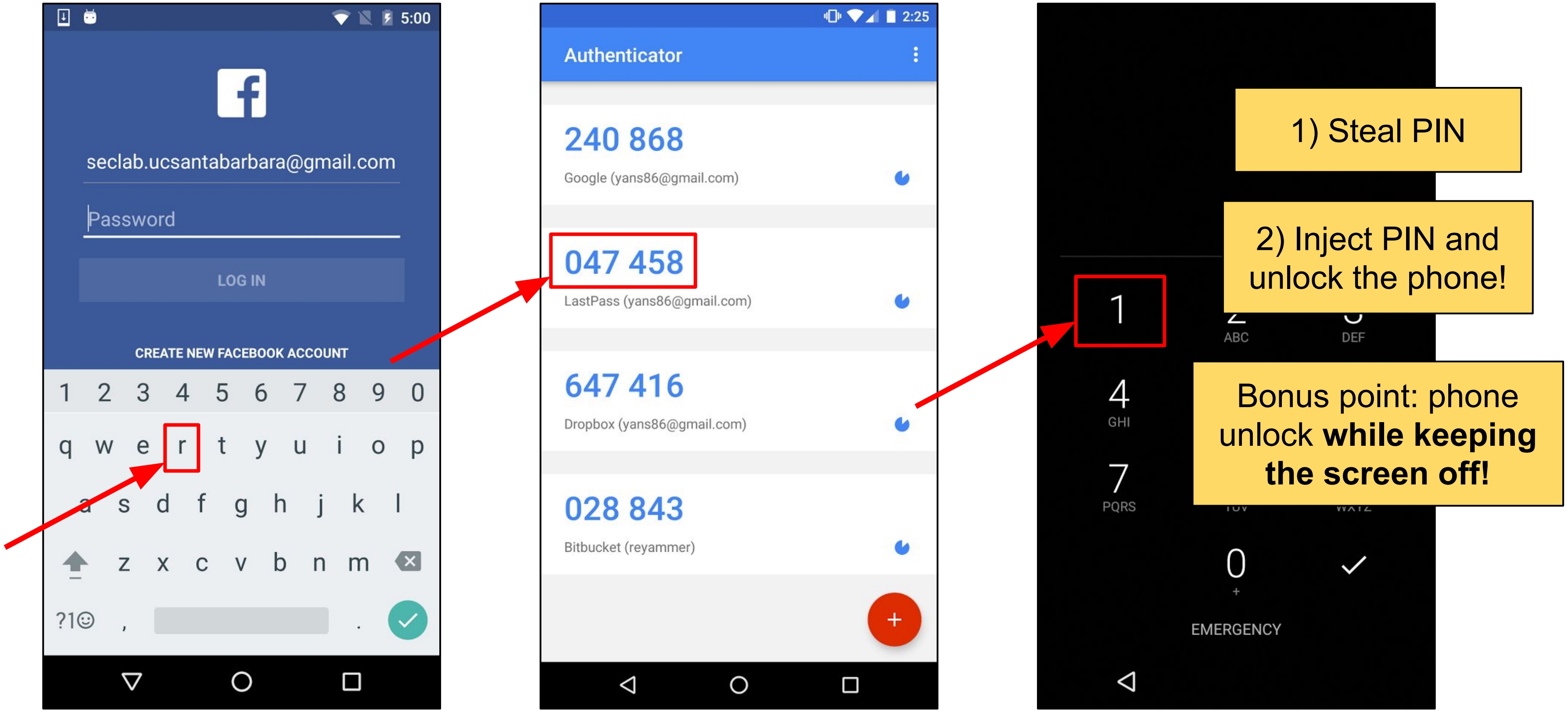
- The inherent complexity of the Android “WindowManager” leads to the creation of unexpected side channels
  - UI security as an afterthought

- Violation of the principle of least privilege
  - Why can an app create *invisible* overlays? *Passthrough* overlays?
  - Overlays are **completely** customizable!

Attack: a11y on steroids

- Yet another design shortcoming:
  - By default, UI objects are all considered non-security sensitive
  - Security should be the rule, not the exception!

Attack: a11y on steroids


5:00
seclab.ucsantabarbara@gmail.com
Password
LOG IN
CREATE NEW FACEBOOK ACCOUNT
2:25
Authenticator
240 868
Google (yans86@gmail.com)
047 458
LastPass (yans86@gmail.com)
647 416
Dropbox (yans86@gmail.com)
028 843
Bitbucket (reyammer)
1
ABC
DEF
4
GHI
7
PQRS
0
EMERGENCY

1) Steal PIN

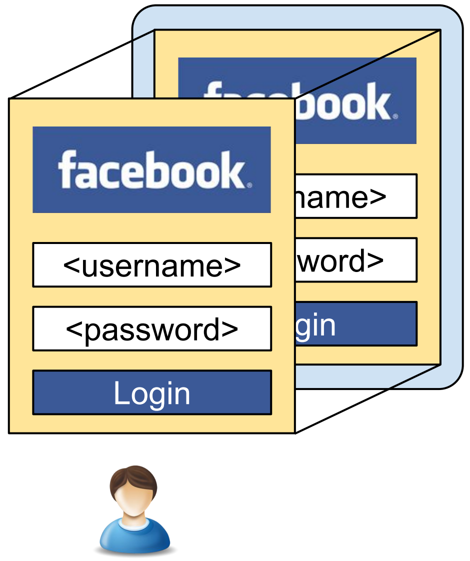
2) Inject PIN and unlock the phone!

Bonus point: phone unlock **while keeping the screen off!**

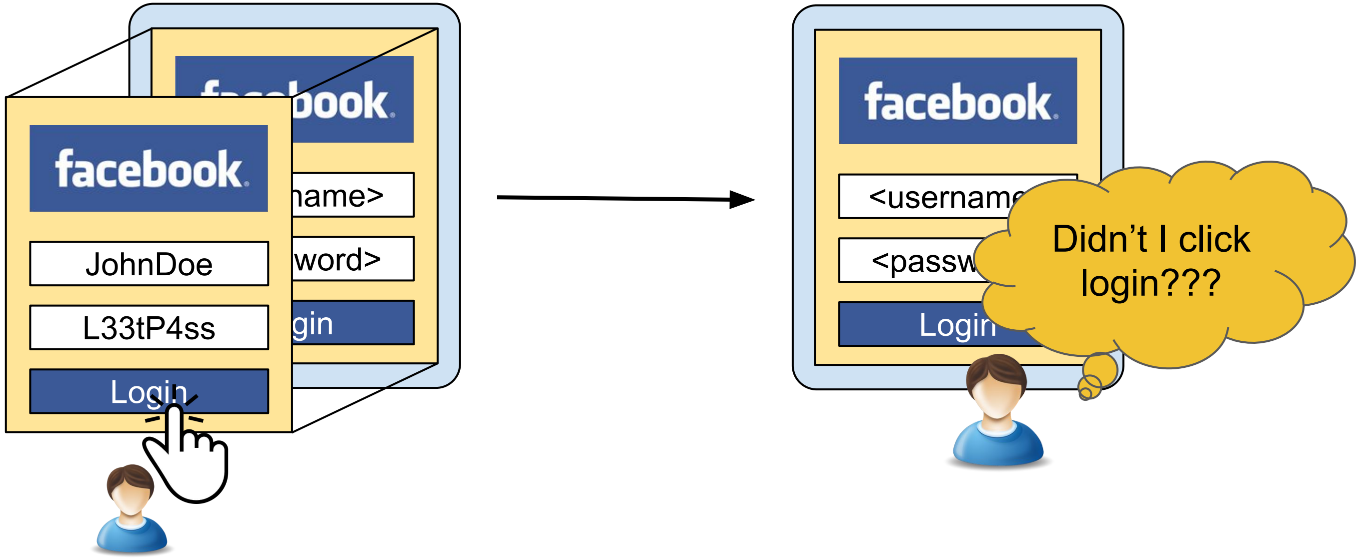
Cloak & Dagger attacks

- You can mount even nastier attacks by combining the two permissions!

Traditional Phishing

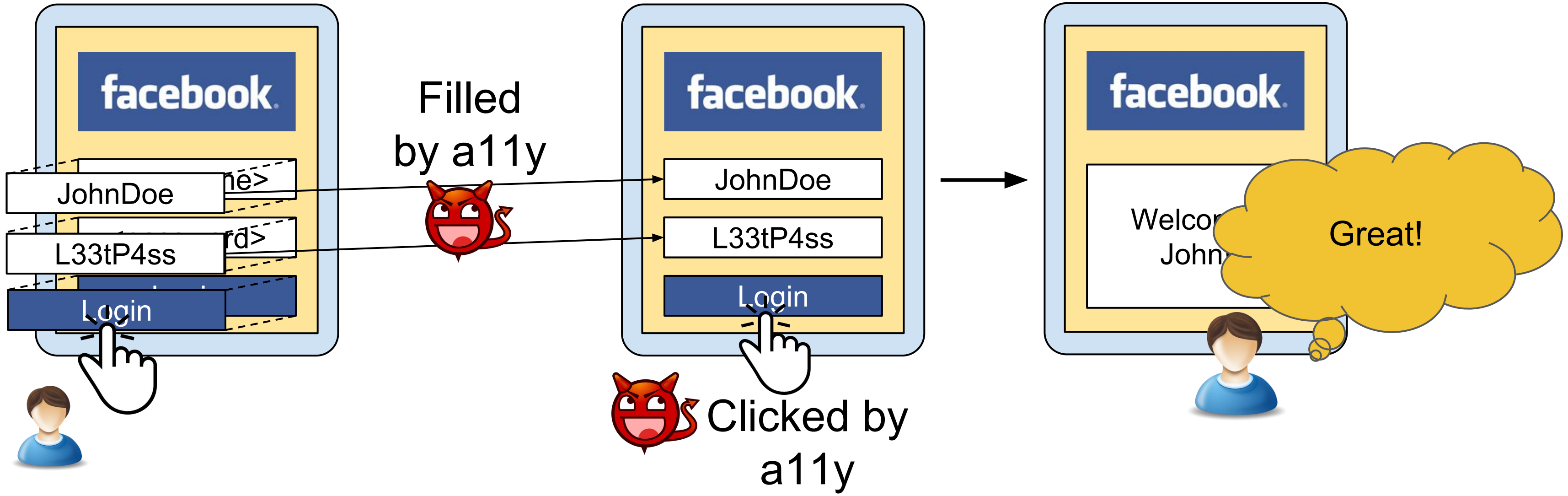


Traditional Phishing



Attack: Stealthy Phishing

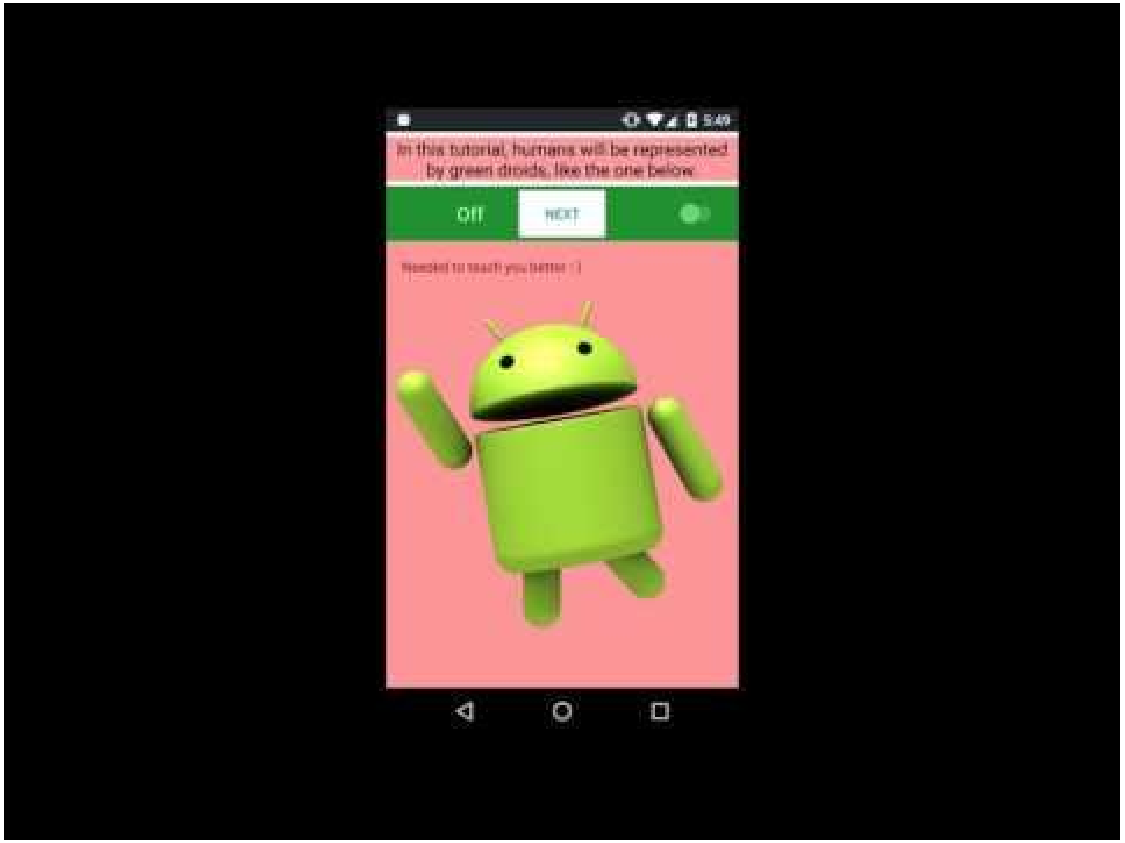
UI-in-the-middle Attack



Attack: Silent God-mode App Installation

- We show a video to the user...
- ...and, behind the scene, we do nasty things via a11y
- The grand plan
  - Silent installation of super-malicious app
  - Enable all its permissions
  - Clean up steps

Clickjacking ~> a11y & Silent God-mode App Install



In this tutorial, humans will be represented by green droids, like the one below

Needed to teach you better :)

The Attacker’s Grand Plan

- Start from app with two permissions
- Install secondary malicious app with all permissions
- The initial malicious app uninstalls itself

Net result

- God-mode app installed on the victim’s phone
- The user has NO idea about what happened

Additional Attack Scenarios

- Advanced ransomware
  - Block device by changing the PIN to an attacker-controlled one
- Covering and clicking around on Chrome
  - Taking over victim’s Google account
  - Steal saved passwords
- Additional key point: even if Google fixes its apps, third-party apps will remain vulnerable to these attacks

Are these attacks actually practical?

User Study

- 20 human subjects (all from Georgia Tech)

- Attacks we tested
  - Clickjacking to enable a11y
  - Silent God-mode App Installation
  - Stealthy Phishing

Experiment Settings

- We let the user play with an app we wrote, twice
- During one of the runs, the user is attacked
- Then, we asked the subject whether she noticed any difference / weirdness

Results

- Clickjacking to enable a11y
  - None of the subject understood what happened
- Silent God-mode App Installation
  - None of the subject understood what happened
- Stealthy Phishing
  - 18 out of 20 did not detect any difference
  - The remaining two triggered a bug in our prototype, and they reported “graphical glitches” (but they did not understand they were attacked)

Overall Awareness

- Do users know about these two permissions?
- Results are worrisome
  - Only 2 out of 20 knew about the “draw on top” permission
  - Only 5 out of 20 knew about a11y
  - ***No subject knew about both!***
- ...why should they look for them?

How can we fix this?

Responsible Disclosure

- “Simple bugs” via AOSP reports (August 22nd, 2016)

Responsible Disclosure

- “Simple bugs” via AOSP reports (August 22nd, 2016)
    - Invisible Grid Attack ~> Moderate severity (not fixed yet)

Responsible Disclosure

- “Simple bugs” via AOSP reports (August 22nd, 2016)
  - Invisible Grid Attack ~> Moderate severity (not fixed yet)
  - A11y on steroids

Responsible Disclosure

- “Simple bugs” via AOSP reports (August 22nd, 2016)
  - Invisible Grid Attack ~> Moderate severity (not fixed yet)
  - A11y on steroids ~> ???

Disclosure of “a11y on steroids” (August 22nd)

- Bug marked as “Won’t fix, work as intended” (September 30th)
- Bug marked as “High severity” (October 18th)
- Downgraded to “Won’t fix” because “limiting those services would render the device unusable” (November 28th)
- “We will update the documentation” (May 4th)
- AND THEY DID!!!11!1!

a11y documentation “patch”

- AccessibilityEvent’s “security note” is silently removed
    - June 6th version vs current version

- “Patch the documentation not the code”.

- 0day in the documentation! Where is my CVE?!

Responsible Disclosure

- “Simple bugs” via AOSP reports (August 22nd, 2016)
    - Invisible Grid Attack ~> Moderate severity
    - A11y on steroids ~> ???

Responsible Disclosure

- “Simple bugs” via AOSP reports (August 22nd, 2016)
  - Invisible Grid Attack ~> Moderate severity
  - A11y on steroids ~> ???
  - New clickjacking technique

Responsible Disclosure

- “Simple bugs’ 22nd, 2016)
  - Invisible Grid Attack ~> Moderate severity

Android Rewards
Qualifying Vulnerabilities

Few classes of vulnerabilities will generally not qualify for a reward:

- **Tap-jacking** and **UI-redressing** attacks that involve tricking the user into tapping a UI element

"BUT, BUT, BUT...."
memegenerator.net

FFFFFFFFFFFFFFFFUUUUUUUUUU

Responsible Disclosure

- “Simple bugs” via AOSP reports (August 22nd, 2016)
    - Invisible Grid Attack ~> Moderate severity
    - A11y on steroids ~> ???
    - New clickjacking technique ~>

Responsible Disclosure

- “Simple bugs” via AOSP reports (August 22nd, 2016)
  - Invisible Grid Attack ~> Moderate severity
  - A11y on steroids ~> ???
  - New clickjacking technique ~> :-(

- Shared the paper draft with Adrian Ludwig, head of Android security (December 19th)

Responsible Disclosure

- "Simple bugs" via AOSP reports (August 22nd, 2016)
  - Invisible Grid Attack ~> Moderate severity
  -
  -

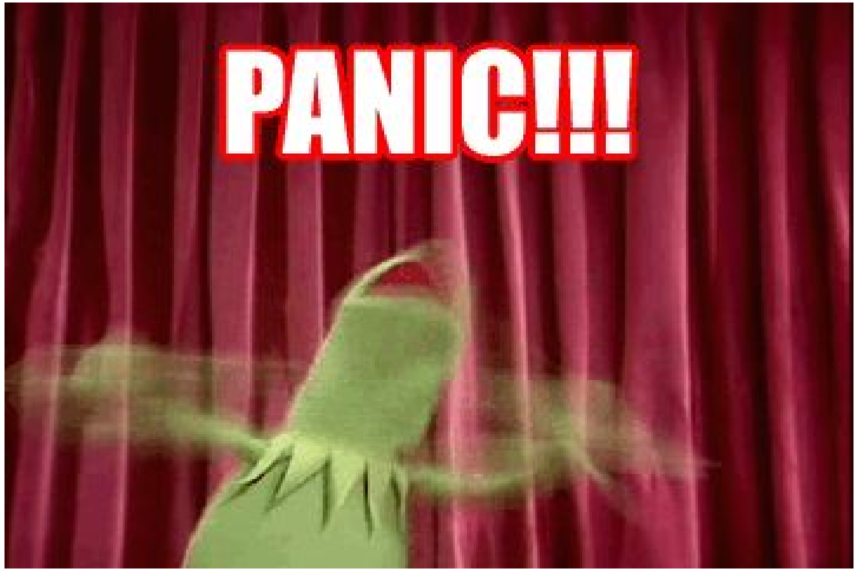
- Sha

  Android security (December 19th)

All attacks are still working!

(Even on Android 7.1.2, with **July**'s updates)

How is the Android security team reacting?



“I’m not alone”

- UI security is not considered a “big deal”

- Check Nick Kralevich’s talk at Android Security Symposium, March 2017 (https://youtu.be/ITL6VHOFQj8?t=57m40s)
  - First question during the Q&A...

“I’m not alone”

- UI security is not considered a “big deal”

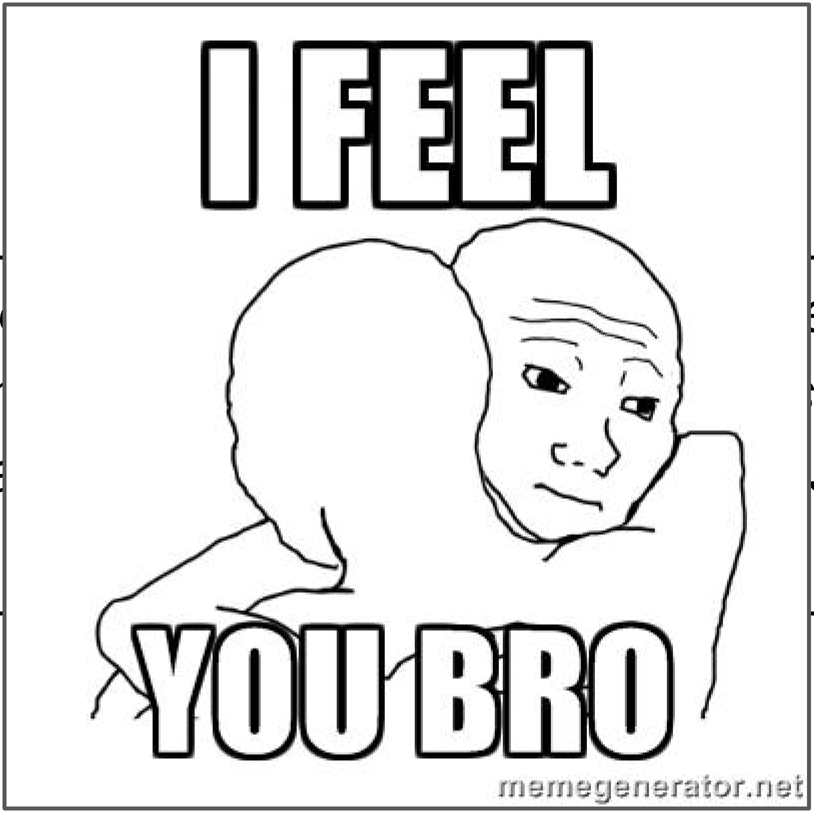
“There are also plain boring bugs, for example in the UI [...], personally I don’t report them anymore because you just don’t care. My bugs are hanging with the ‘new’ status for years then they are just auto-closed.”

“I’m not alone”

- UI security is

-

“There are als   *e in the UI [...], personally I dor*   *e you just don’t care. My bugs a*   *s for years then*

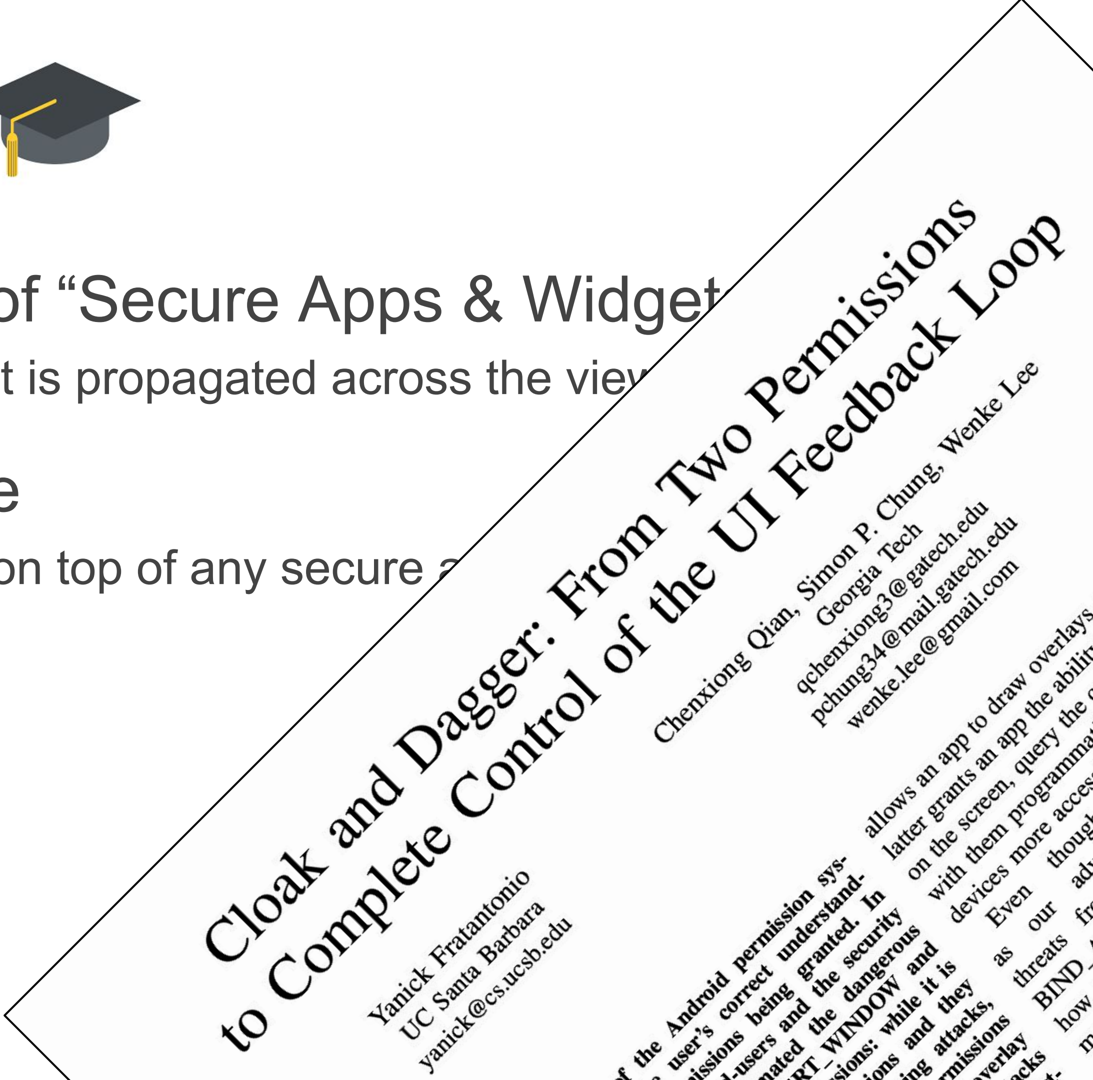


Securing Android UI

- Introduce the concept of “Secure Apps & Widgets”
  - Defined through a flag that is propagated across the view tree
- OS-enforced guarantee
  - No overlay will be shown on top of any secure app/widget
- System popups
  - Inspired by web popups

Securing Android UI

- Introduce the concept of “Secure Apps & Widget
  - Defined through a flag that is propagated across the vie
- OS-enforced guarantee
  - No overlay will be shown on top of any secure a
- System popups
  - Inspired by web popups



Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop

Yanick Fratantonio
UC Santa Barbara
yanick@cs.ucsb.edu

Chenxiong Qian, Simon P. Chung, Wenke Lee
Georgia Tech
qchenxiong3@gatech.edu
pchung34@mail.gatech.edu
wenke.lee@gmail.com

of the Android permission sys-
user's correct understand-
issions being granted. In
-users and the security
ated the dangerous
RT_WINDOW and
sions: while it is
ons and they
ng attacks,
missions
verlay
acks

allows an app to draw overlays
latter grants an app the abilit
on the screen, query the
with them programmat
devices more acces
Even thoug
our adv
as fr
threats
BIND_
how

Short-term Recommendations

- “Draw on top” permission should ***not*** be automatically granted!
    - When manually enabling it, the user will at least have a chance to learn what this permission can actually do

- More thorough vetting process for apps requiring both “draw on top” and a11y
    - They are not many, manual vetting could be feasible

What happened next...

- Work presented at IEEE Security & Privacy 2017
    - Distinguished Practical Paper award!
- We setup a website, cloak-and-dagger.org
- A lot of press coverage...

Google’s official answer

“*[...] We have updated Google Play Protect — our security services on all Android devices with Google Play — to detect and prevent the installation of these apps. Prior to this report, we had already built new security protections into Android O that will further strengthen our protection from these issues moving forward.*”

Detect Cloak & Dagger

- What we would do:
  - Detect apps that combine these two permissions

- Does the attacker really need both permissions?

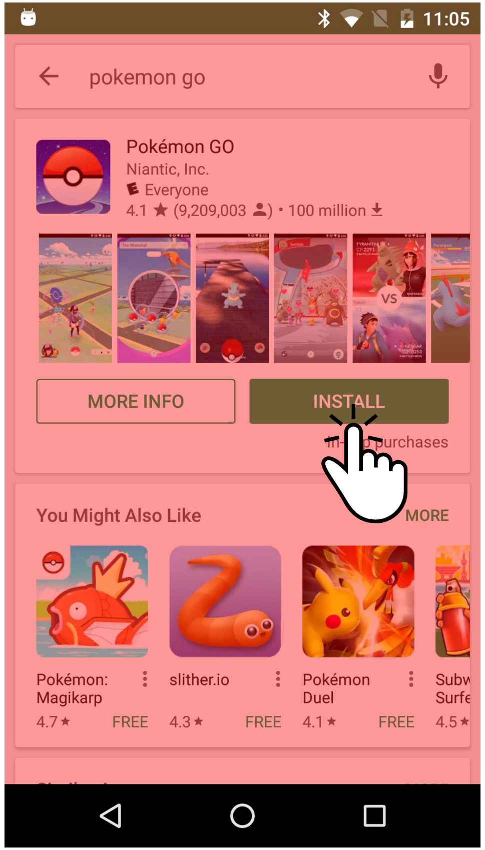
- NOPE!

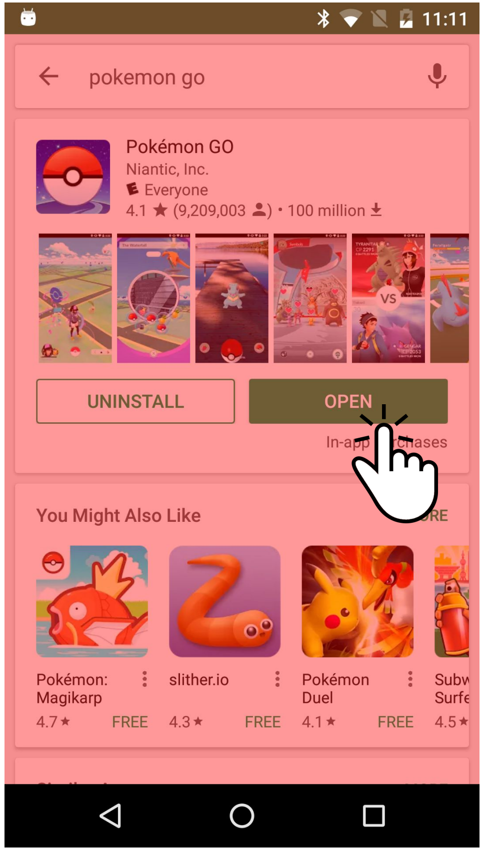
Bootstrap the attacks from one permission

- Start with an app that only requires "SYSTEM_ALERT_WINDOW"
- Install a secondary malicious app that only requires a11y!
- How?

CLICKJACKING
EVERYWHERE
memegenerator.net

Clickjacking Everywhere!





Google Play Store app is vulnerable to clickjacking as well :-(

Start the Play Store app programmatically via Intents

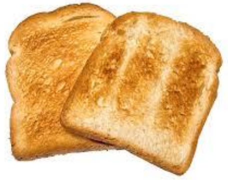
Let’s go one step further...

- ...do we actually need the SYSTEM_ALERT_WINDOW?

Let’s go one step further...

- SYSTEM_ALERT WINDOW permission is needed to create windows of “TYPE_ALERT_SYSTEM”

- Realization: the attacker just needs to create windows on top of all apps’ activities
    - She does **not** need to go over “system” windows (e.g., status bar, navigation bar)
    - Any overlay’s “type” that goes on top of activities is enough



FTW!

- Toasts are usually created with this API:
  - `makeText(Context context, int resId, int duration)`
  - Duration: either 2 seconds or 3.5 seconds
  - Limited customization capabilities

- It is possible to create arbitrarily custom “Toasts”
  - TYPE_SYSTEM_ALERT ~> TYPE_TOAST
  - “Pretty simple” to port all the attacks
    - `sed -i “s/TYPE_SYSTEM_ALERT/TYPE_TOAST/” *`

Impact & Caveats

- Android 6.0.1
  - You can bootstrap Cloak & Dagger attacks with **zero** permissions
  - Caveat: you need to steal two more clicks to install the app with a11y
- Android 7.1.2
  - Several mechanisms against Toast abuse
  - The SYSTEM_ALERT_WINDOW permission is required

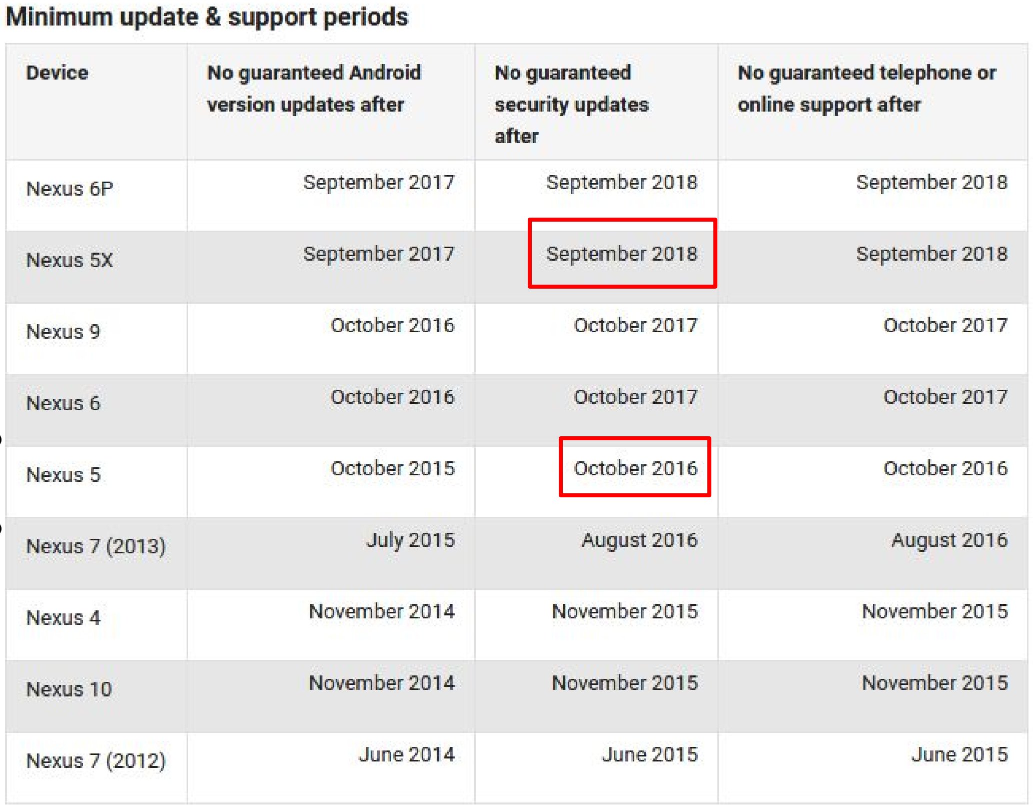
Existing work on Toasts for clickjacking

- Blog post “Revisiting Android Tapjacking”
  - It discusses how to use Toasts for clickjacking to enable permissions

"We also hope that Google will begin to make use of their own security features in future releases of their own packages, such as settings, dialer, and market applications."

Date: May 24th **2011**!

Current state of Android security updates

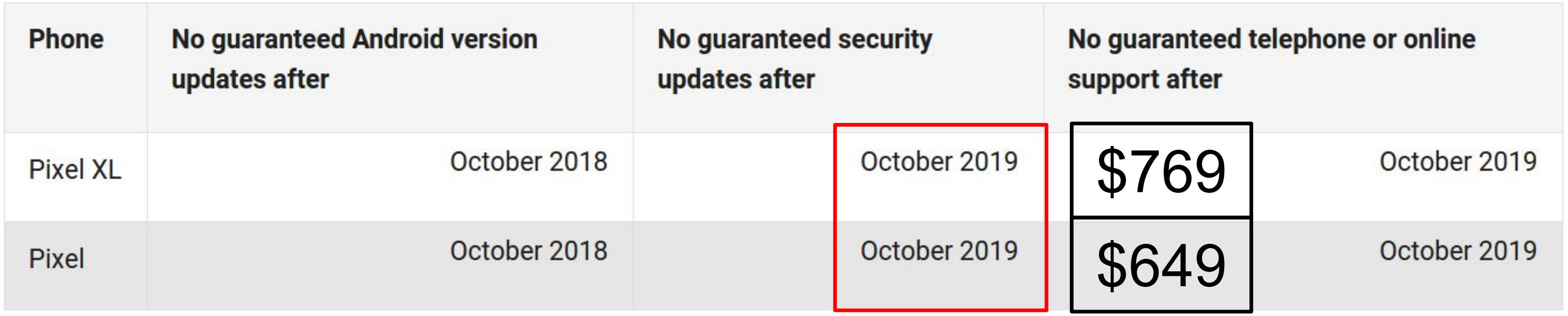


Minimum update & support periods

Device	No guaranteed Android version updates after	No guaranteed security updates after	No guaranteed telephone or online support after
Nexus 6P	September 2017	September 2018	September 2018
Nexus 5X	September 2017	September 2018	September 2018
Nexus 9	October 2016	October 2017	October 2017
Nexus 6	October 2016	October 2017	October 2017
Nexus 5	October 2015	October 2016	October 2016
Nexus 7 (2013)	July 2015	August 2016	August 2016
Nexus 4	November 2014	November 2015	November 2015
Nexus 10	November 2014	November 2015	November 2015
Nexus 7 (2012)	June 2014	June 2015	June 2015

Stuck with Android 6.0.1

Current state of Android security updates



Phone	No guaranteed Android version updates after	No guaranteed security updates after	No guaranteed telephone or online support after
Pixel XL	October 2018	October 2019	$769 October 2019
Pixel	October 2018	October 2019	$649 October 2019

Android O (Preview 3 developer version)

- Invisible Grid Attack is fixed! YEAH!
- Clickjacking: currently more vulnerable than before
  - The final “OK” button to enable a11y is NOT protected by the obscured flag :-(
- “A11y on steroids” attacks “work as intended” ;-)

Call to action for Google

- Take boring UI bugs seriously. They may be boring at first, but when put together they can bypass all low-level security features.

- Security updates situation is bad, even for “Google devices” (like Nexus and Pixel lines).
    - Monthly updates are ~~> *great* <~~, but...
    - Google Pixel is $649, end of support is October 2019 :-(

ion

rin . T

wh ey c

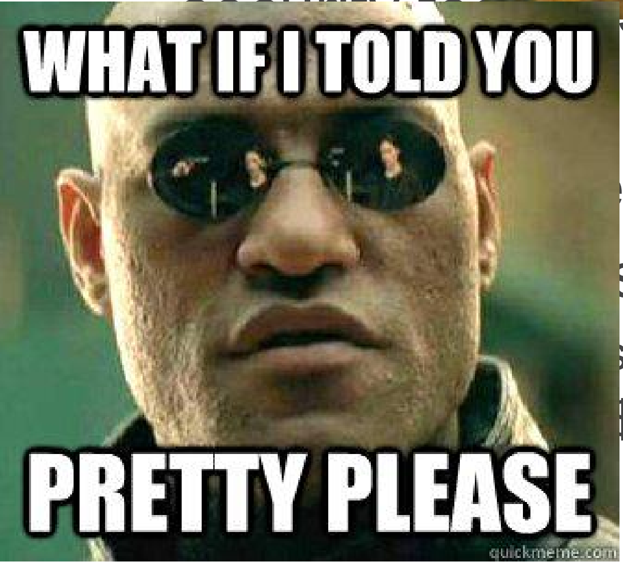
security fo

es e

s' K

s a

$6 b



Takeaways

Yanick Fratantonio
@reyammer
https://cs.ucsb.edu/~yanick
yanick@cs.ucsb.edu

- “Cloak & Dagger” attacks
  - Stealthy, powerful, and practical UI attacks
  - New ways to abuse known “problematic” UI “features”
- UI security bugs matter
  - They are the low-hanging fruits for the attackers
- More info: cloak-and-dagger.org