

Android as a Research Platform

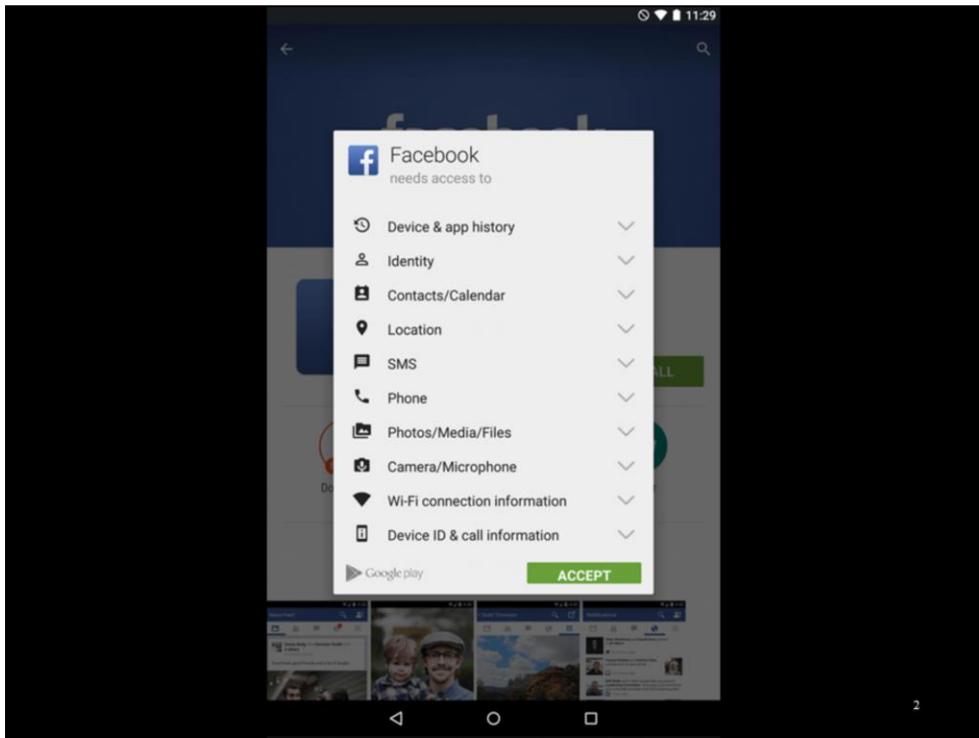
Irwin Reyes



Previous a researcher in the usable security group at ICSI/UCB

Now a lead research engineer at Two Six Labs working on DARPA Brandeis

This talk goes over work from UCB



Install-time permissions

Accept all or don't use the app at all. No obvious hints for why an app is requesting a certain privilege. Can be overwhelming to end-users.

Android Permissions Remystified: A Field Study on Contextual Integrity

Primal Wijesekera¹, Arjun Baokar², Ashkan Hosseini², Serge Egelman²,
David Wagner², and Konstantin Beznosov¹

¹*University of British Columbia, Vancouver, Canada,*

{primal,beznosov}@ece.ubc.ca

²*University of California, Berkeley, Berkeley, USA,*
{arjunbaokar,ashkan}@berkeley.edu, {egelman,daw}@cs.berkeley.edu

Abstract

We instrumented the Android platform to collect data regarding how often and under what circumstances smartphone applications access protected resources regulated by permissions. We performed a 36-person field study to explore the notion of “contextual integrity,” i.e., how often applications access protected resources when users are not expecting it. Based on our collection of 27M data points and exit interviews with participants, we examine the situations in which users would like the ability to deny applications access to protected resources. At least 80% of our participants would have preferred to prevent at least one permission request, and overall, they stated a desire to block over a third of all requests. Our findings pave the way for future systems to automatically determine the situations in which users would want to be confronted with security decisions.

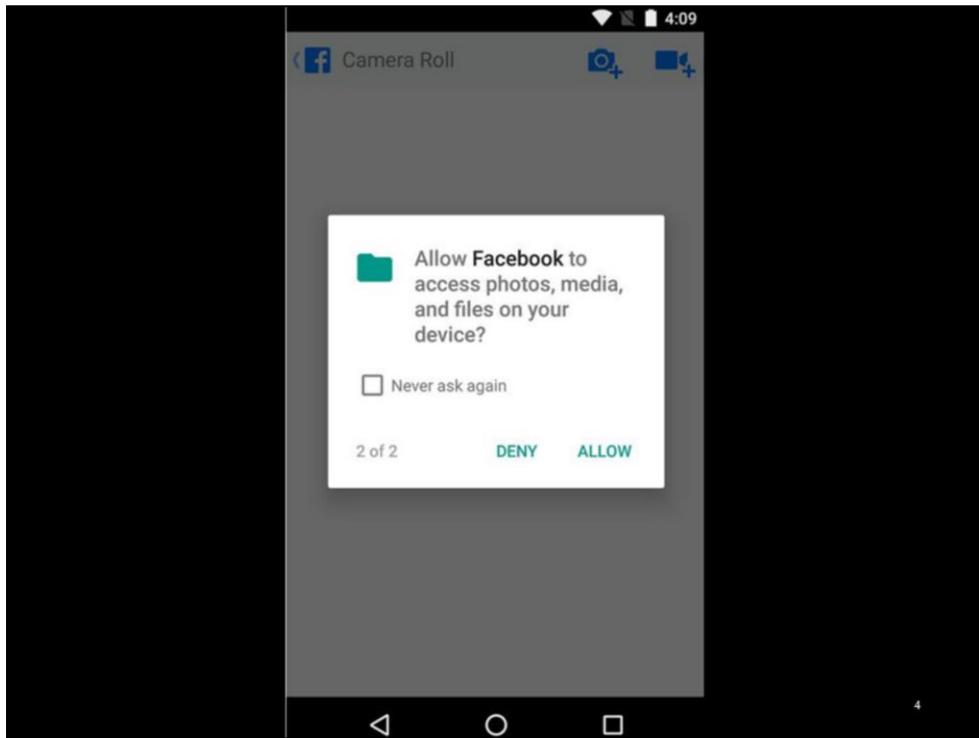
time the data is actually requested, it is not clear whether or not users are being prompted about access to data that they actually find concerning, or whether they would approve of subsequent requests [15].

Nissenbaum posited that the reason why most privacy models fail to predict violations is that they fail to consider contextual integrity [32]. That is, privacy violations occur when personal information is used in ways that defy users’ expectations. We believe that this notion of “privacy as contextual integrity” can be applied to smartphone permission systems to yield more effective permissions by only prompting users when an application’s access to sensitive data is likely to defy expectations. As a first step down this path, we examined how applications are currently accessing this data and then examined whether or not it complied with users’ expectations.

3

I joined the usable security group at UC Berkeley shortly after they published results showing over 1 in 3 attempts to access sensitive data are unwanted by the user under the install-time model.

This work motivated further research into better aligning permissions systems with user privacy preferences.



Run-time permissions were introduced to Android in version 6 “Marshmallow,” released in October 2015. Asks for permission on the first time an app tries to access the protected resource (i.e., “ask-on-first-use” or AOFU).

An improvement to install-time permissions. This provides contextual clues to the user: in this example, Facebook needs to read photos and videos for its Camera Roll feature.

Uber begins background collection of rider location data

Kate Conger



@kateconger / 7:08 pm EST • November 28, 2016

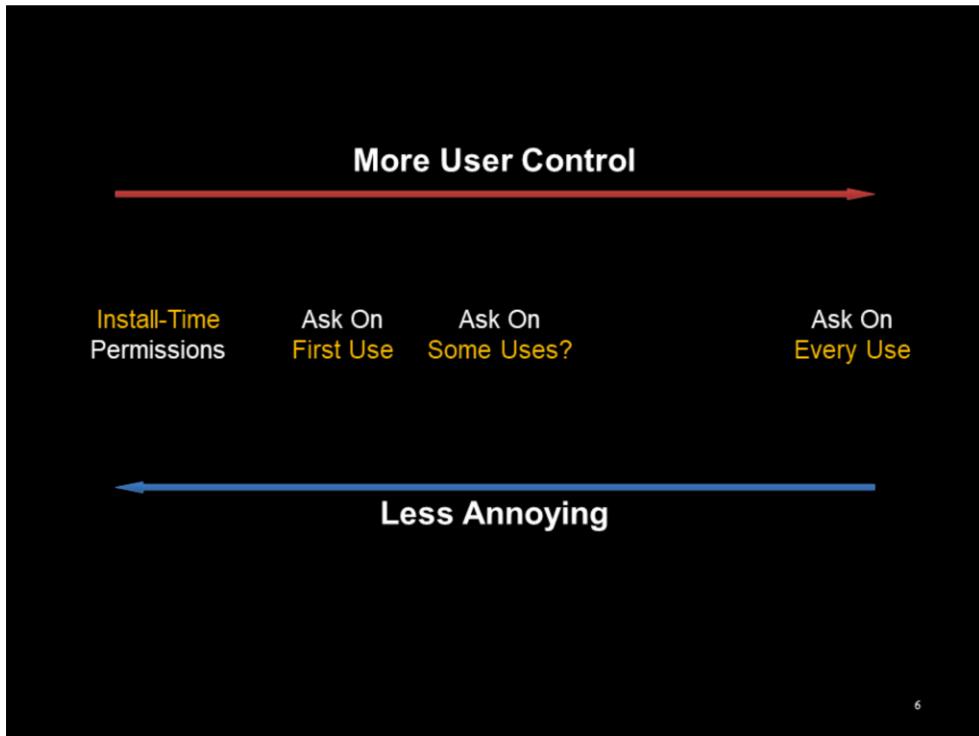
Comment



5

AOFU is an improvement, but only captures user privacy preferences in one context: the first time a permission is exercised. It naively applies that decision to all future contexts.

A user might be OK with Uber collecting their information in requesting a ride, but not for continuous location tracking.



AOFU has shortcomings, so how can we improve it?

We can naively ask on every use. But unusable.

How about we ask on some uses? Let's prototype evaluate it. This requires modifying Android.

OBLIGATORY iOS DISCLAIMER



7

Why is a lot of mobile security/privacy research on Android? What about iOS?

Closed source: Can't modify.

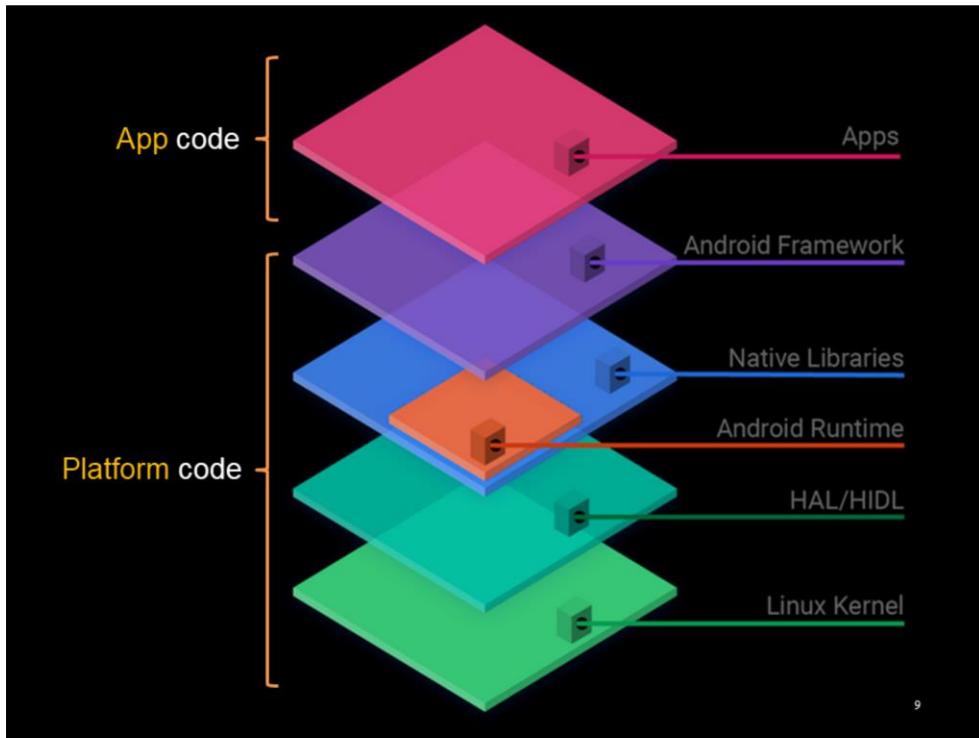
Encrypted app packages (.ipa files): Requires jailbroken phone to decrypt; lots of hoops to jump through.

However, can still install root cert to MITM traffic; only mildly annoying to do.



8

Going forward, all methods and results in this talk are most relevant to Android.



Android apps operate at the top of the software stack: Apps call functions exposed by the Android framework. For example, functions to manipulate on-screen UI elements or read sensitive user data (e.g., location, contact information, etc.).

The Android framework is the highest level of abstraction, acting as a front-end to the underlying software/hardware stack. This makes it easy to write one app that works for a broad set of Android devices.

source Set up Design More ▾ Search LANGUAGE ▾ GO TO CODE SIGN IN

Android unites the world! Use the open source Android operating system to power your device. GET SOURCE



Interfaces and architecture
Learn how the pieces fit together, from the kernel to the HALs to updatable system components.
[UNDERSTAND ARCHITECTURE](#)

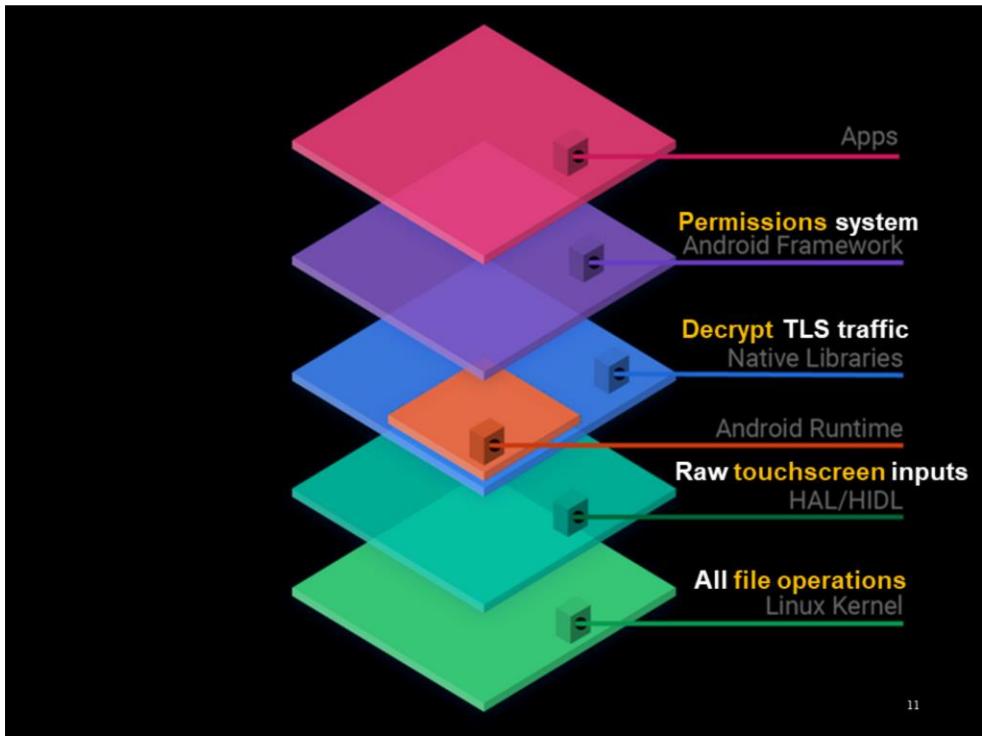
Securing Android is essential
Find out how the Android security program works and learn how to implement the latest features.
[IMPLEMENT SECURITY](#)

Design compatible devices
Offer a consistent experience with other Android-powered devices for users and app developers.
[TEST DEVICES](#)

<https://source.android.com>

10

Just as you can write Android apps, you can write your own fork of the Android platform too.



11

Some cool things you can do up and down the stack:

- Framework: Custom permissions system
- Native libraries: Capture unencrypted TLS traffic
- HAL/HIDL: Get raw touchscreen input data
- Linux kernel: Log all file operations

HIGH LEVEL DESCRIPTIONS AHEAD



<https://source.android.com>

12

Modifying, deploying, and testing Android source code has a lot of little quirks and details associated with it. Will only go over how to get started with it. More detailed documentation at source.android.com

My goal is to give you enough to be curious and ask questions, so feel free to contact me. It took me a while to get comfortable with it myself.

Twitter: [@irwinreyes.com](https://twitter.com/@irwinreyes)

Email: irwin.reyes@twosixlabs.com OR ioreyes@icsi.berkeley.edu OR
email@irwinreyes.com

What you need to develop Android



Linux build environment



Ample hard drive space



Multi-core CPU



Android smartphone



Lots of time and patience

13

To build Android, you'll need:

1. Modern Linux build environment. Ubuntu Server 19 LTS generally works out of the box. Might need to install gcc and openjdk.
2. Lots of hard disk space. The Android 9 source tree takes up about 150 GBs. Compiling it for a phone will result in about 250 GBs of output.
3. Building Android can be done in parallel. More CPU cores = faster (but is eventually disk-bound).
4. A smartphone compatible with the version of `Android you're developing. Nexus 5/5X/6P recommended for Android 6 through 8. Pixel series recommended for Android 9 and 10.
Can develop using VMs, but VMs are slow and unreliable.
5. Building Android can take a long time. Debugging is done by using log lines. No runtime debugger for the OS. Long turnaround between building and installing on phones.

```

1 [||||| | 97.5%] 21 [||| | 100.0%] 41 [||| | 99.4%] 61 [||||| | 100.0%]
2 [||||| | 100.0%] 22 [||| | 100.0%] 42 [||| | 99.4%] 62 [||||| | 100.0%]
3 [||||| | 100.0%] 23 [||| | 98.1%] 43 [||| | 100.0%] 63 [||||| | 100.0%]
4 [||||| | 100.0%] 24 [||| | 100.0%] 44 [||| | 100.0%] 64 [||||| | 100.0%]
5 [||||| | 100.0%] 25 [||| | 100.0%] 45 [||| | 100.0%] 65 [||||| | 100.0%]
6 [||||| | 100.0%] 26 [||| | 100.0%] 46 [||| | 100.0%] 66 [||||| | 100.0%]
7 [||||| | 98.7%] 27 [||| | 100.0%] 47 [||| | 100.0%] 67 [||||| | 100.0%]
8 [||||| | 98.1%] 28 [||| | 100.0%] 48 [||| | 100.0%] 68 [||||| | 100.0%]
9 [||||| | 98.7%] 29 [||| | 100.0%] 49 [||| | 100.0%] 69 [||||| | 100.0%]
10 [||||| | 100.0%] 30 [||| | 99.4%] 50 [||| | 100.0%] 70 [||||| | 100.0%]
11 [||||| | 100.0%] 31 [||| | 98.7%] 51 [||| | 100.0%] 71 [||||| | 100.0%]
12 [||||| | 100.0%] 32 [||| | 100.0%] 52 [||| | 100.0%] 72 [||||| | 100.0%]
13 [||||| | 100.0%] 33 [||| | 100.0%] 53 [||| | 100.0%] 73 [||||| | 100.0%]
14 [||||| | 100.0%] 34 [||| | 100.0%] 54 [||| | 100.0%] 74 [||||| | 100.0%]
15 [||||| | 100.0%] 35 [||| | 100.0%] 55 [||| | 100.0%] 75 [||||| | 100.0%]
16 [||||| | 100.0%] 36 [||| | 100.0%] 56 [||| | 100.0%] 76 [||||| | 100.0%]
17 [||||| | 99.4%] 37 [||| | 99.4%] 57 [||| | 99.4%] 77 [||||| | 100.0%]
18 [||||| | 100.0%] 38 [||| | 100.0%] 58 [||| | 100.0%] 78 [||||| | 100.0%]
19 [||||| | 100.0%] 39 [||| | 100.0%] 59 [||| | 100.0%] 79 [||||| | 98.7%]
20 [||||| | 100.0%] 40 [||| | 100.0%] 60 [||| | 100.0%] 80 [||||| | 97.5%]
Mem[||||| | 18.0G/188G] Tasks: 489, 442 thr; 80 running
Swp[ | 23.0M/976M] Load average: 135.95 80.95 35.17
Uptime: 23 days, 00:02:10

```

~25 minutes to build Android!

14

On a 40-core (80 logical) server with SSDs, building Android from scratch takes about 25 minutes.

Luckily, you can do incremental builds afterwards.

The `repo` tool

Google Git [Code Review](#) [Generate Password](#) [Revoke Passwords](#) [Sign in](#)

Git repositories on android

Name	Description
accessories/manifest	
api_council_filter	Parent for API additions that requires Android API Council approval. OBSOLETE: API-Review is now defined in A...
assets/android-studio-ux-assets	Bug: 32992167
brillo/manifest	
cts_drcn_filter	Parent project for CTS projects that requires Dr.No +2's.
device/aeon/upboard	
device/amlogic/yukawa	Bug: 122486287
device/amlogic/yukawa-kernel	Bug: 122486287
device/asus/deb	
device/asus/flo	
device/asus/flo-kernel	
device/asus/fugu	

15

The Android source tree is made up of several hundred Git repositories. The “repo” tool manages those Git projects; initialize build environment, pull code, check for outstanding changes, etc.

Each Git project roughly corresponds to a particular part of Android: device-specific code, the Linux kernel, preinstalled apps, etc.

The `repo` tool

Build	Tag	Version	Supported devices	Security patch level
QP1A.191005.007.A1	android-10.0.0_r5	Android 10	Pixel 2 XL, Pixel 2, Pixel XL, Pixel	2019-10-06
QP1A.191005.007	android-10.0.0_r4	Android 10	Pixel 3a XL, Pixel 3a, Pixel 3 XL, Pixel 3	2019-10-05
QP1A.190711.020.C3	android-10.0.0_r3	Android 10	Pixel 3a XL, Pixel 3a, Pixel 3 XL, Pixel 3	2019-09-05
QP1A.190711.020	android-10.0.0_r2	Android 10	Pixel 3a XL, Pixel 3a, Pixel 3 XL, Pixel 3, Pixel 2 XL, Pixel 2, Pixel XL, Pixel	2019-09-05
QP1A.190711.019	android-10.0.0_r1	Android 10	Pixel 3a XL, Pixel 3a, Pixel 3 XL, Pixel 3, Pixel 2 XL, Pixel 2, Pixel XL, Pixel	2019-09-05

16

The `repo` tool can also manage branches and tags. When first initializing the build environment, you have to pick a tag corresponding to the version of Android you want and what device you're targeting.

As mentioned before, Pixel phones are highly recommended for modern Android development. Older releases target all the Nexus phones.

Compiling Android in 7 steps

1. Get the `repo` tool

<https://source.android.com/setup/build/downloading>

2. Create working directory and initialize to target branch

```
> mkdir aosp; cd aosp  
> repo init -u https://android.googlesource.com/platform/manifest  
-b android-10.0.0_r2
```

3. Download source tree

```
> repo sync -j$(nproc) | ⏺
```

17

Use the repo tool to select the Android version. Version tags available at <https://source.android.com/setup/start/build-numbers>

Compiling Android in 7 steps

4. Set environment variables and functions

```
> source build/envsetup.sh
```

5. Use the `lunch` command to select target device

```
> lunch aosp_taimen-userdebug
```

6. Compile Android

```
> make -j$(nproc)
```



7. Get Android system image files and flash onto phone

```
> cp out/target/product/aosp_taimen-userdebug/*.img $DESTDIR  
> cd $DESTDIR  
> fastboot --disable-verification flash vbmeta vbmeta.img  
> fastboot erase system; fastboot system system.img
```

18

Use the lunch tool to select the target device. Target device codenames available at <https://developers.google.com/android/images> Taimen is the codename for the Pixel 2XL.

Fastboot commands assume the phone bootloader has already been unlocked. How to do this is left as an exercise to the reader.

Incremental builds only need the steps on this slide. 4 – 7 for a new session, and only 6&7 for an existing session.

Android platform development tips

AndroidXRef

ANDROID SOURCE

- Pie - 9.0.0_r3
- Oreo - 8.1.0_r33
- Oreo - 8.0.0_r4
- Nougat - 7.1.2_r36
- Nougat - 7.1.1_r6
- Nougat - 7.0.0_r1
- Marshmallow - 6.0.1_r10
- Marshmallow - 6.0.0_r5
- Marshmallow - 6.0.0_r1
- Lollipop - 5.1.1_r6
- Lollipop - 5.1.0_r1
- Lollipop - 5.0.0_r2
- KitKat - 4.4.4_r1
- KitKat - 4.4.3_r1.1
- KitKat - 4.4.2_r2
- KitKat - 4.4.2_r1
- KitKat - 4.4

News

- 2018-08-11 - New Index: Pie - 9.0.0_r3
- 2018-06-19 - New Index: Oreo - 8.1.0_r33
- 2017-12-22 - New Index: Nougat - 7.1.2_r36
- 2017-09-19 - New Index: Oreo - 8.0.0_r4
- 2016-12-22 - New Index: Nougat - 7.1.1_r6
- 2016-08-24 - New Index: Nougat - 7.0.0_r1
- 2016-01-09 - New Index: Marshmallow - 6.0.1_r10
- 2016-01-07 - New Index: Marshmallow - 6.0.0_r5
- 2015-10-07 - New Index: Marshmallow - 6.0.0_r1
- 2015-07-03 - New Index: Lollipop - 5.1.1_r6
- 2015-03-13 - New Index: Lollipop - 5.1.0_r1
- 2014-09-06 - New Index: Lollipop - 5.0.0_r2
- 2014-07-14 - New Index: KitKat - 4.4.4_r1
- 2014-06-06 - New Index: KitKat - 4.4.3_r1.1
- 2014-03-25 - New Index: KitKat - 4.4.2_r2
- 2014-01-08 - New Index: KitKat - 4.4.2_r1
- 2013-11-01 - New Index: KitKat - 4.4
- 2013-10-31 - New Index: Gingerbread - 2.3.7

KERNEL SOURCE

- Kernel - 3.18
- Kernel - 3.14
- Kernel - 3.10
- Kernel - 3.4
- Kernel - 3.3
- Kernel - 3.0
- Kernel - 2.6.39



<https://androidxref.com>

19

The source tree is huge and hard to navigate. Android Xref is a really useful resource for searching through the code.

Android platform development tips



Name	Date	Size
..	10-Aug-2018	4 KIB
Android.bp	10-Aug-2018	47 KIB
Android.mk	10-Aug-2018	43.4 KIB
apct-tests/	10-Aug-2018	4 KIB
api/	10-Aug-2018	4 KIB
CleanSpec.mk	10-Aug-2018	24.4 KIB
cmds/	10-Aug-2018	4 KIB
config/	10-Aug-2018	4 KIB
core/	10-Aug-2018	4 KIB
data/	10-Aug-2018	4 KIB
docs/	10-Aug-2018	4 KIB
drm/	10-Aug-2018	4 KIB
graphics/	10-Aug-2018	4 KIB
keystore/	10-Aug-2018	4 KIB
libs/	10-Aug-2018	4 KIB
location/	10-Aug-2018	4 KIB
lowpan/	10-Aug-2018	4 KIB
media/	10-Aug-2018	4 KIB
MODULE_LICENSE_APACHE2	10-Aug-2018	0
native/	10-Aug-2018	4 KIB

Start looking in frameworks/base/

20

Most modifications will touch the frameworks/base project. This is where nearly all API functions used by apps are implemented.

Useful terminology: Managers are app-space code that are front-ends to system-space ManagerServices that actually talk to the underlying HAL.

For example, LocationManager (app-space) and LocationManagerService (system-service implementation).

Android XRef Pie 9.0.0_r3

Cross Reference: **PermissionManagerService.java**

xref: /frameworks/base/services/core/java/com/android/server/pm/permission/PermissionManagerService.java

Home | History | Annotate | Line# | Navigate | Download only in PermissionManagerService.java

```
85 /**
86  * Manages all permissions and handles permissions related tasks.
87 */
88 public class PermissionManagerService {
89     private static final String TAG = "PackageManager";
90
91     /** Permission grant: not grant the permission. */
92     private static final int GRANT_DENIED = 1;
93     /** Permission grant: grant the permission as an install permission. */
94     private static final int GRANT_INSTALL = 2;
95     /** Permission grant: grant the permission as a runtime one. */
96     private static final int GRANT_RUNTIME = 3;
97     /** Permission grant: grant as runtime a permission that was granted as an install time one. */
98     private static final int GRANT_UPGRADE = 4;
99
100    /** Cap the size of permission trees that 3rd party apps can define; in characters of text */
101    private static final int MAX_PERMISSION_TREE_FOOTPRINT = 32768;
102    /** Empty array to avoid allocations */
103    private static final int[] EMPTY_INT_ARRAY = new int[0];
104}
```

21

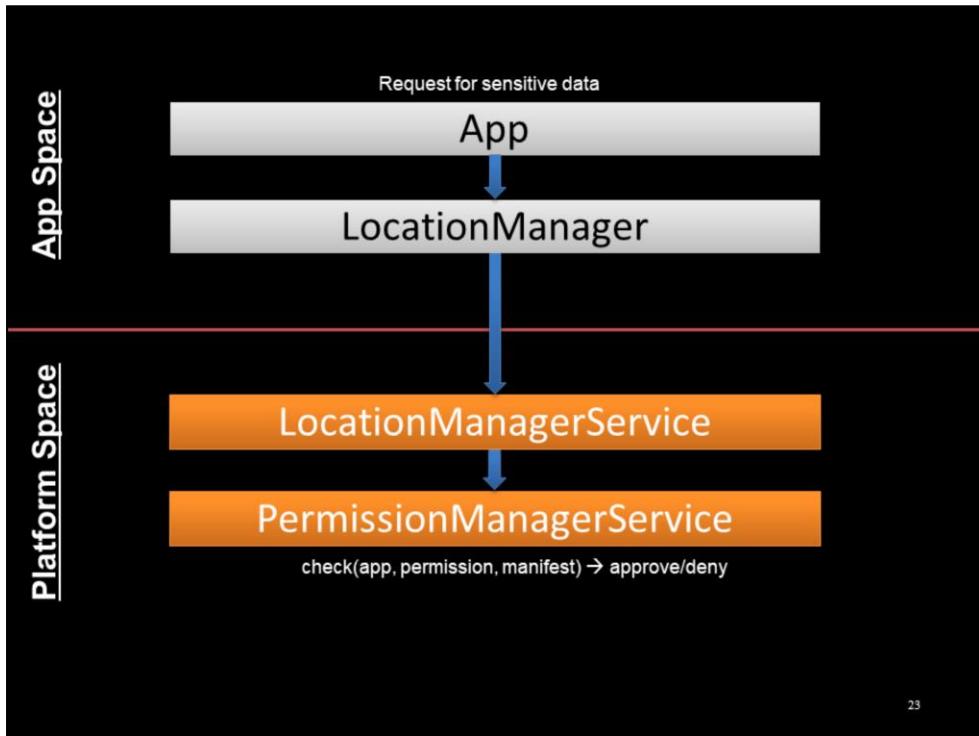
Coincidentally, frameworks/base/ also has a PermissionManagerService.

“Manages all permissions and handles permissions related tasks.” Hmm...

**we modified the PermissionManagerService
to use context and predict user preferences**

22

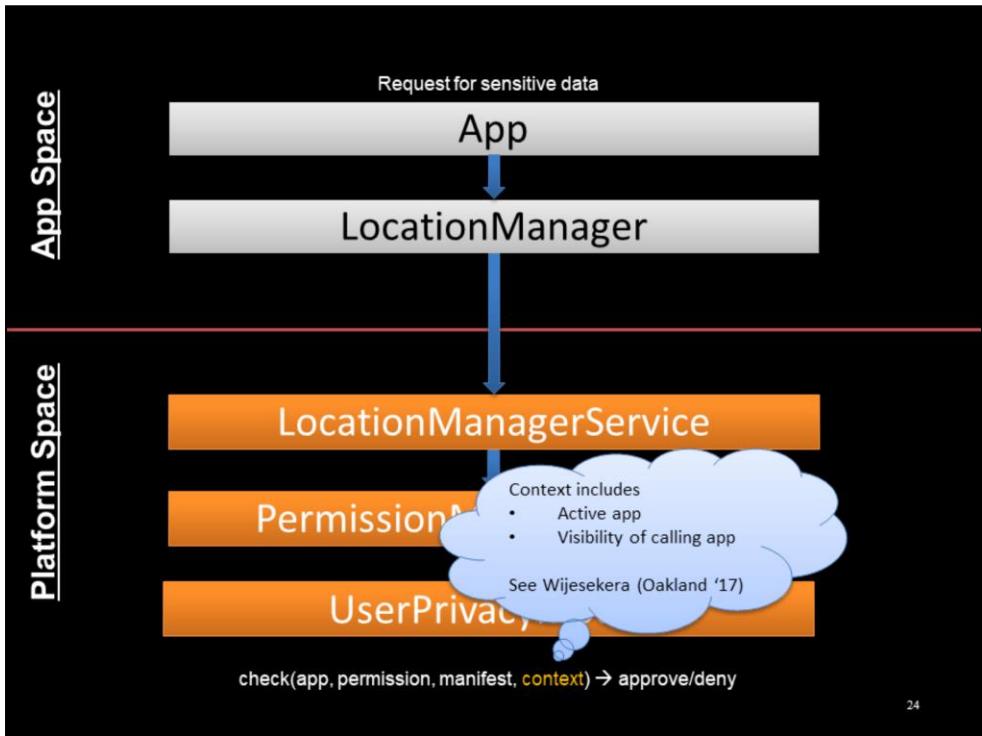
Permission requests go through the PermissionManagerService.



Normally, when an app requests sensitive data (e.g., location), it goes through the corresponding manager.

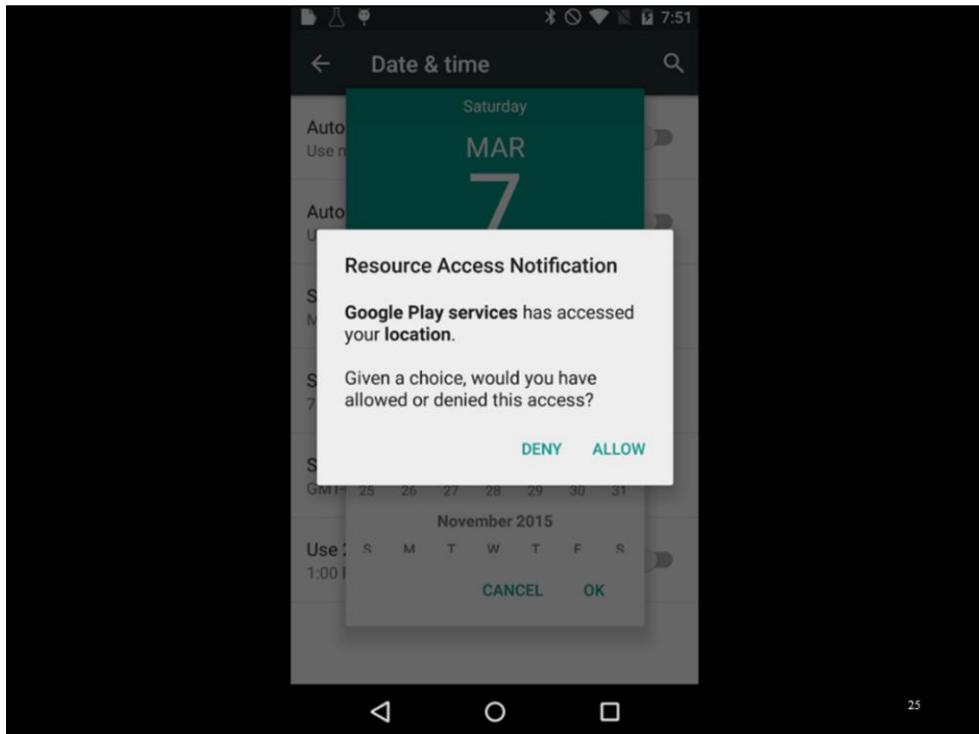
The manager talks to the backing service, which requests a permission check.

The PermissionManagerService checks if the app has declared the appropriate permission in the manifest and if the user has approved it under AOFU.
Approves the access if so.



We modified this flow to include context in the request, which is used by an additional step called by the PermissionManagerService:

The context is used to predict user preferences based on a prebuilt bootstrapped classifier model. It has a training phase for personalization. See Oakland paper for more details.



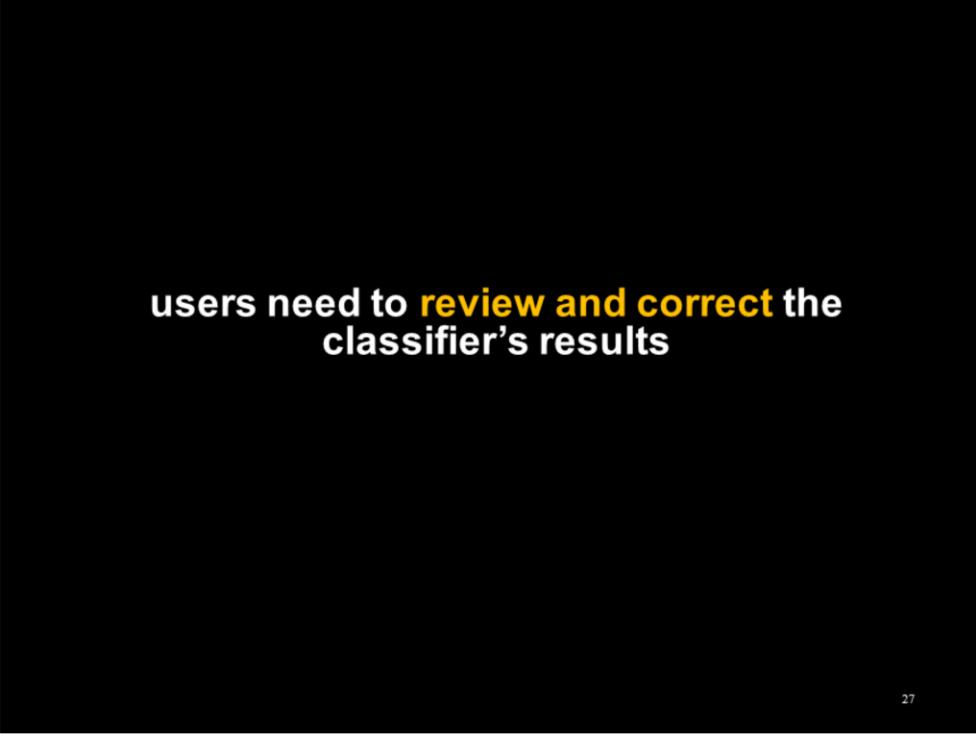
In practice, this works very similarly to the existing AOFU model.

But the user is prompted when either the classifier is in training mode (i.e., when device is first used) or when the classifier produces low-confidence results.

**the classifier model can produce
incorrect or unwanted results sometimes**

26

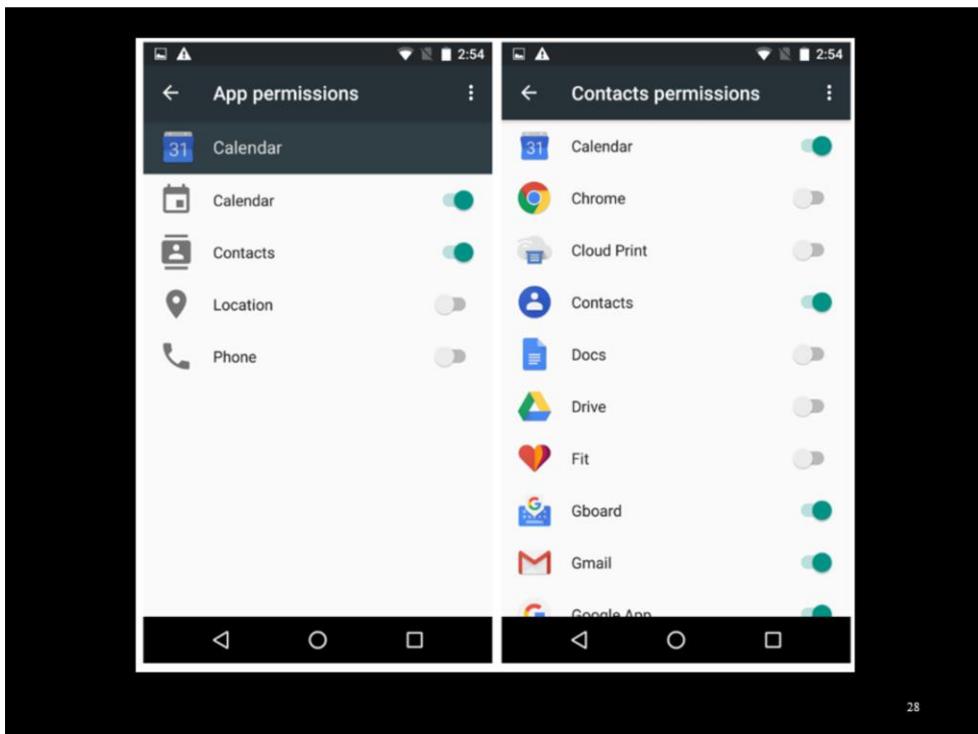
In practice though, the classifier isn't perfect. It will still produce unwanted outcomes sometimes.



**users need to review and correct the
classifier's results**

27

How can users control this without being overwhelmed?



28

Existing configuration tools for permissions are insufficient: They only offer blanket on/off toggles, and they don't give any information about the circumstances in which permissions were exercised.

TurtleGuard: Helping Android Users Apply Contextual Privacy Preferences

Lynn Tsai¹, Primal Wijesekera², Joel Reardon¹, Irwin Reyes³, Jung-Wei Chen⁴, Nathan Good⁴, Serge Egelman^{1,3}, and David Wagner¹

¹University of California, Berkeley, Berkeley, CA
E-mail address: Chakravarthy@cs.berkeley.edu

{lynntsai,jreardon}@berkeley.edu, daw@cs.berkeley.edu

²University of British Columbia, Vancouver, BC primal@ece.ubc.ca ³International Computer Science Institute, Berkeley, CA {ioreyes,egelman}@icsi.berkeley.edu

⁴Good Research, Inc., El Cerrito, CA
jennifer,nathan}@goodresearch.com

Contextualizing Privacy Decisions for Better Prediction (and Protection)

Primal Wijesekera¹, Joel Reardon², Irwin Reyes³, Lynn Tsai⁴, Jung-Wei Chen⁵, Nathan Good⁵, David Wagner⁴, Konstantin Beznosov¹, and Serge Egelman^{3,4}

¹University of British Columbia, Vancouver, BC

²University of Calgary, Calgary, AB

onal Computer Science Institute, Berkeley, CA

⁴University of California, Berkeley, CA
⁵Graduate Record Examinations

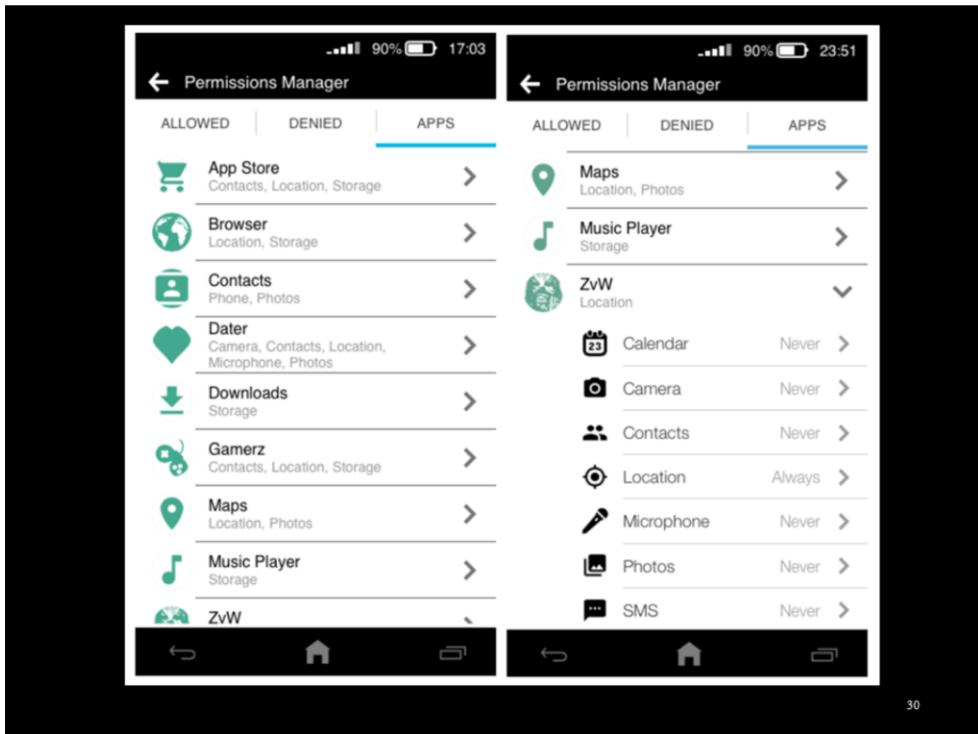
⁵Good Research, Berkeley, CA

bc.ca, joel.reardon@ucalgary.ca

lynntsai@berkeley.edu, {jennifer,nathan}@goodresearch.com, {daw,egelman}@cs.berkeley.edu

29

We developed a front-end configuration tool to support users in contextual permissions systems and tested them.



In the initial TurtleGuard study, we iterated through designs for these controls and evaluated interactive mock-ups of them with 598 participants. 580 produced complete responses, from which the results were drawn.

The final design looked something like this: Have a history of all recently allowed/denied permissions, plus per-app settings.

TASK	CORRECT	INCORRECT
Recent location access	Control: 82.6% Experimental: 82.5%	Control: 17.4% Experimental: 17.5%
Finding granted permissions	Control: 77.1% Experimental: 80.8%	Control: 22.9% Experimental: 19.2%
Background location access	Control: 37.6% Experimental: 78.5%	Control: 62.4% Experimental: 21.5%
Restrict background access	Control: 27.5% Experimental: 76.5%	Control: 72.5% Experimental: 23.5%

31

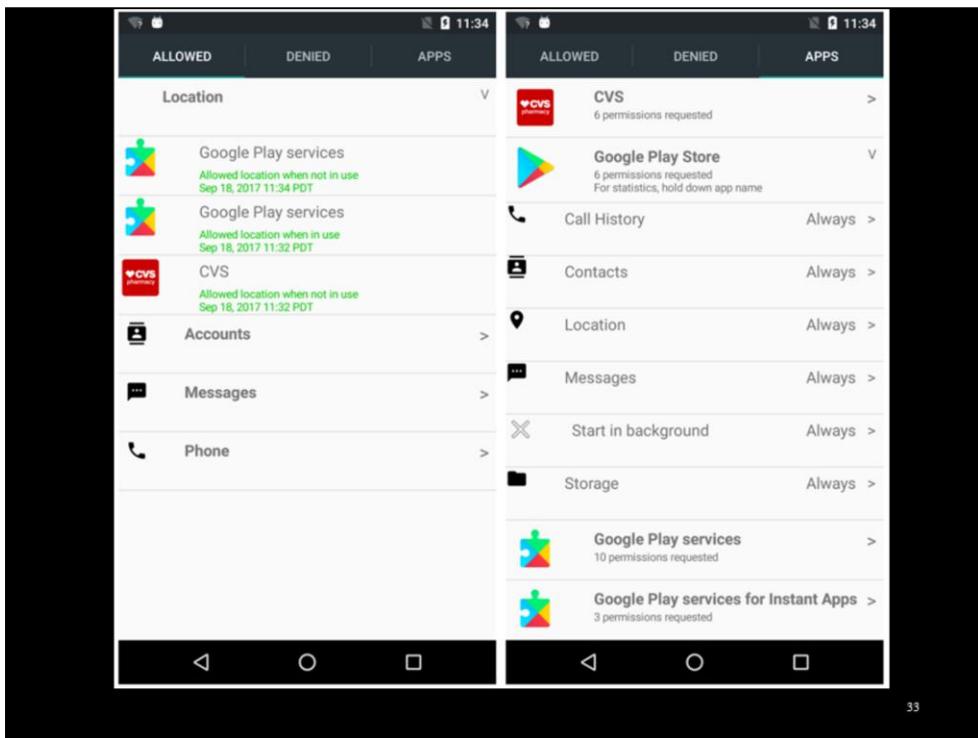
In evaluating these designs, we split the participants into a control group (presented with the stock settings) and an experimental group (presented with TurtleGuard).

Four tasks:

1. Determine the app that most recently accessed location
2. Determine what permissions are granted to a given app
3. Determine if a given app could access location in the background
4. Prohibit app from accessing location in the background

Tasks 3 and 4 take context (app visibility) into account. TurtleGuard fares much better.

**are dynamic permissions and controls
actually usable on real mobile devices?**



The TurtleGuard study steps us through the design of the controls.

We eventually implemented them into the Android platform as part of the system settings. We also implemented a live permissions model for this to control.

field testing with real users

37 participants from Bay Area

1 week using custom Android

usage/error logging and exit interviews

	STOCK ANDROID	DYNAMIC PERMISSIONS
Median permission error rate	Under AOFU: 20.0%	Classifier: 5.26%
Median permission prompts	Under AOFU: 15	Classifier: 13
SecurityExceptions thrown	N/A	Classifier: 3/day (out of 2000 denials/day)

8% expressed annoyance at the prompts

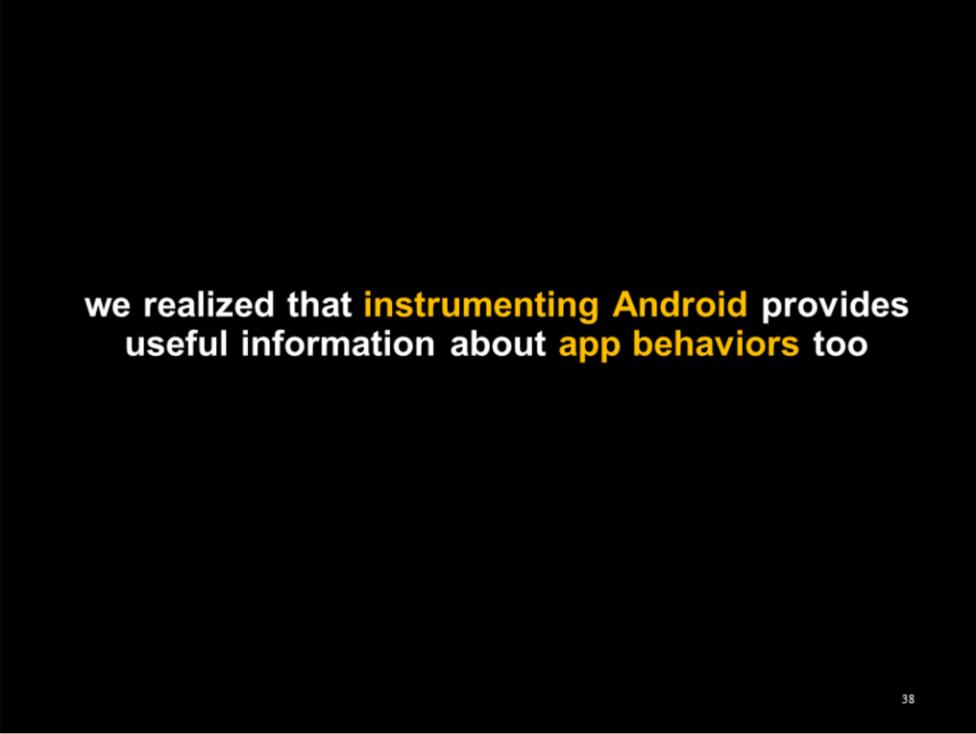
70% surprised at the frequency of app access to sensitive data

40% opened the permission manager (“TurtleGuard”)

GET_ACCOUNTS permission most likely to be restricted by users

READ_CONTACTS permission most likely to be permitted by users

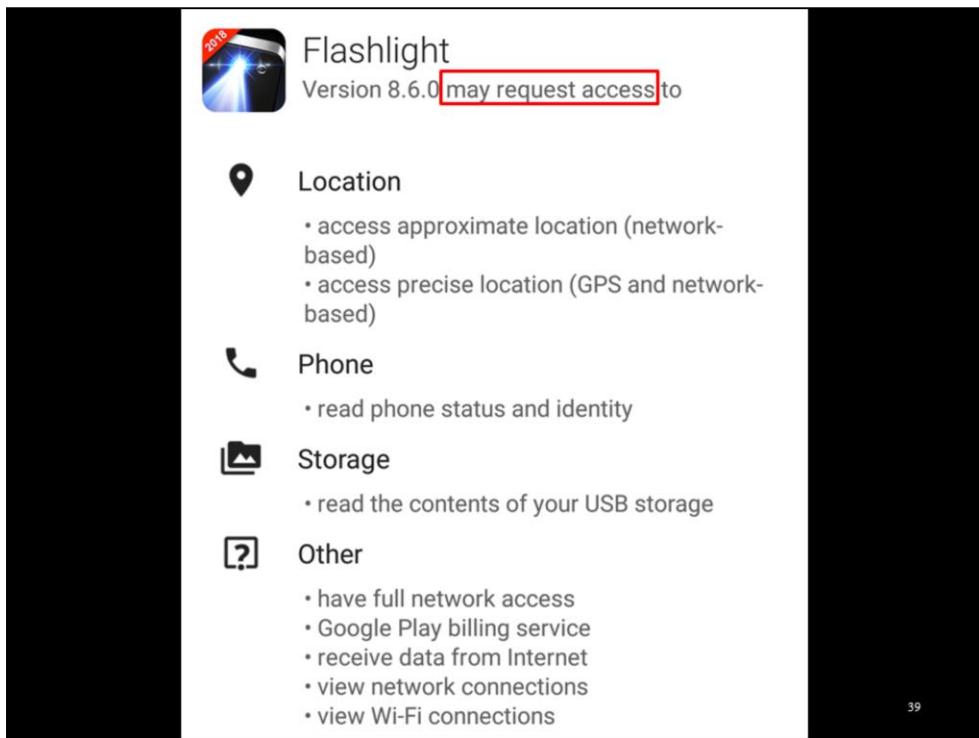
**personal note: realistic user studies are
very time consuming!**



we realized that **instrumenting Android** provides
useful information about **app behaviors** too

38

Because we owned the operating system, we had a very privileged view on how apps interact with user data.



Apps are able to request access to private user data and sensitive device resources.

In their app store listings (such as this one from the Google Play Store), apps disclose their capabilities. However, these disclosures don't tell the full story. Do apps actually use these privileges? With whom do they share sensitive data?

Irwin Reyes*, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, and Serge Egelman

"Won't Somebody Think of the Children?" Examining COPPA Compliance at Scale

Abstract: We present a scalable dynamic analysis framework that allows for the automatic evaluation of the privacy behaviors of Android apps. We use our system to analyze mobile apps' compliance with the Children's Online Privacy Protection Act (COPPA), one of the few stringent privacy laws in the U.S. Based on our automated analysis of 5,855 of the most popular free children's apps, we found that a majority are potentially in violation of COPPA, mainly due to their use of third-party SDKs. While many of these SDKs offer configuration options to respect COPPA by disabling tracking and behavioral advertising, our data suggest that a majority of apps either do not make use of these options or incorrectly propagate them across mediation SDKs. Worse, we observed that 19% of children's apps collect identifiers or other personally identifiable information (PII) via SDKs whose terms of service outright prohibit their use in child-directed apps. Finally, we show that efforts by Google to limit tracking through the use of a resettable advertising ID have had little success: of the 3,454 apps that share the resettable ID with advertisers, 66% transmit other, non-resettable, persistent identifiers as well, negating any intended privacy-preserving properties of the advertising ID.

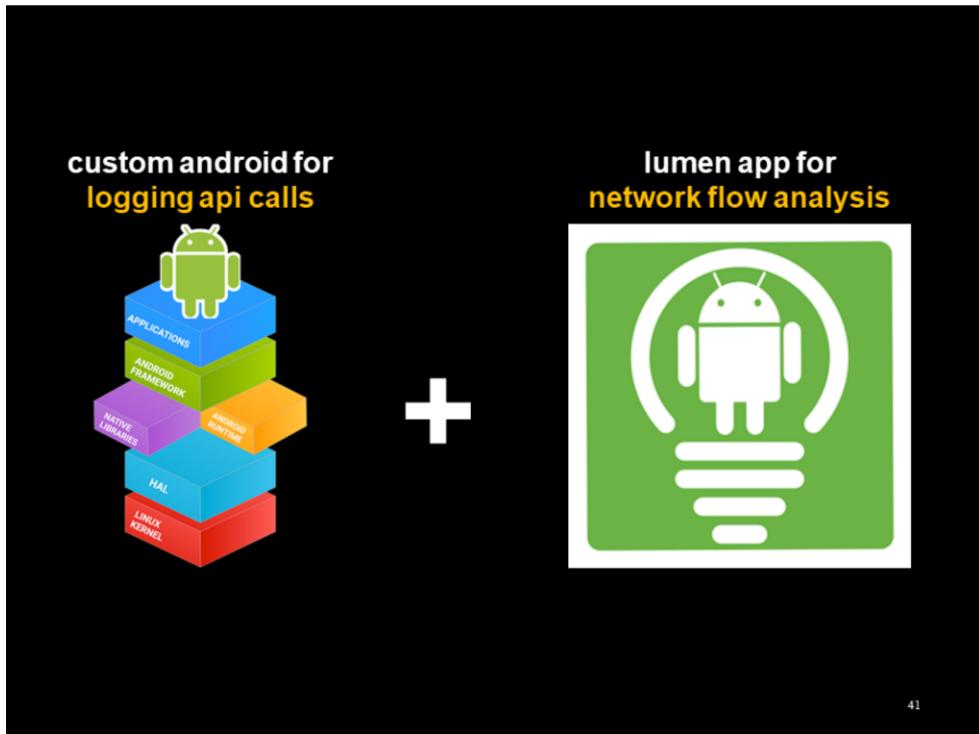
1 Introduction

In the United States, there are few comprehensive privacy regulations. However, one notable exception is the Children's Online Privacy Protection Act (COPPA), which regulates how mobile apps, games and websites are allowed to collect and process personal information from children under the age of 13 [22]. COPPA outright prohibits certain data collection practices, and requires parental consent for others. Of course, enforcement is a painstaking process, as investigations generally rely on manual examination of programs and websites to observe violations [83]. In this paper, we apply our Android dynamic analysis framework to automate the process of detecting potential COPPA violations.

Most current approaches to detecting suspicious application activity on mobile platforms rely on static analysis [*e.g.*, 33, 41, 48, 93] or dynamic analysis [*e.g.*, 28]. However, previous approaches fall short because they either do not observe actual violations, and instead only detect when program *might* contain violative code (in the case of static analysis), or do not scale (in the case of prior dynamic analysis approaches).

We developed a fully automated platform to analyze how apps actually collect and share sensitive data.

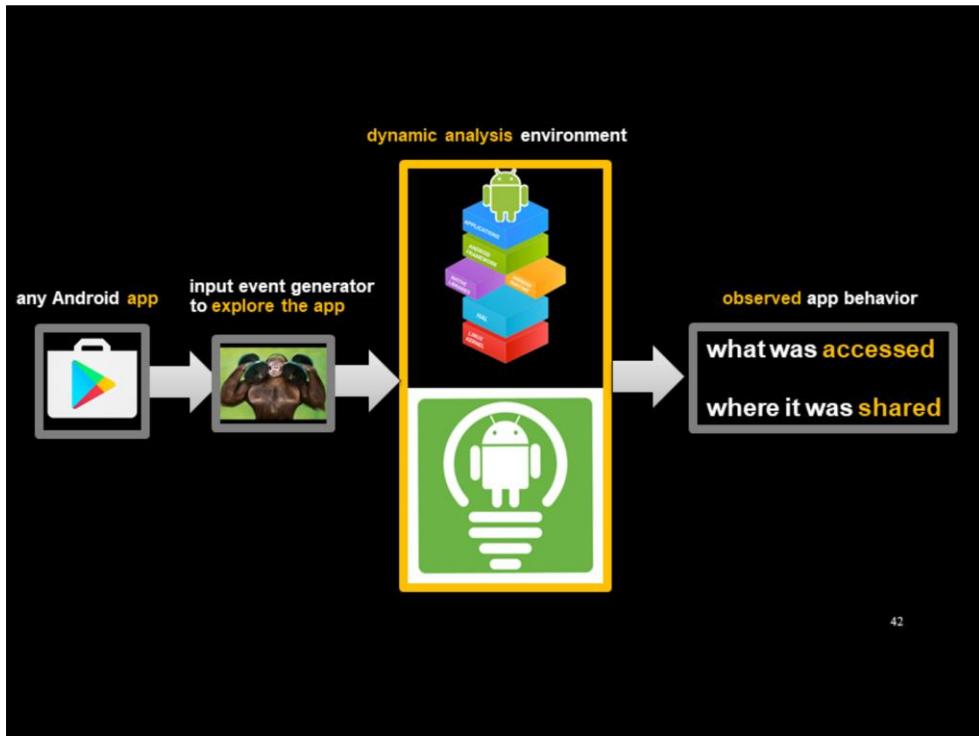
We instrumented the Android operating system and used advanced network traffic monitoring tools. Apps are run and evaluated without any human interaction. Technical details in the paper.



41

Custom Android 6 ROM for observing access to sensitive resources.

Lumen Privacy Monitor to see who gets that info.



42

We run any Android app in this environment and observe its behavior.

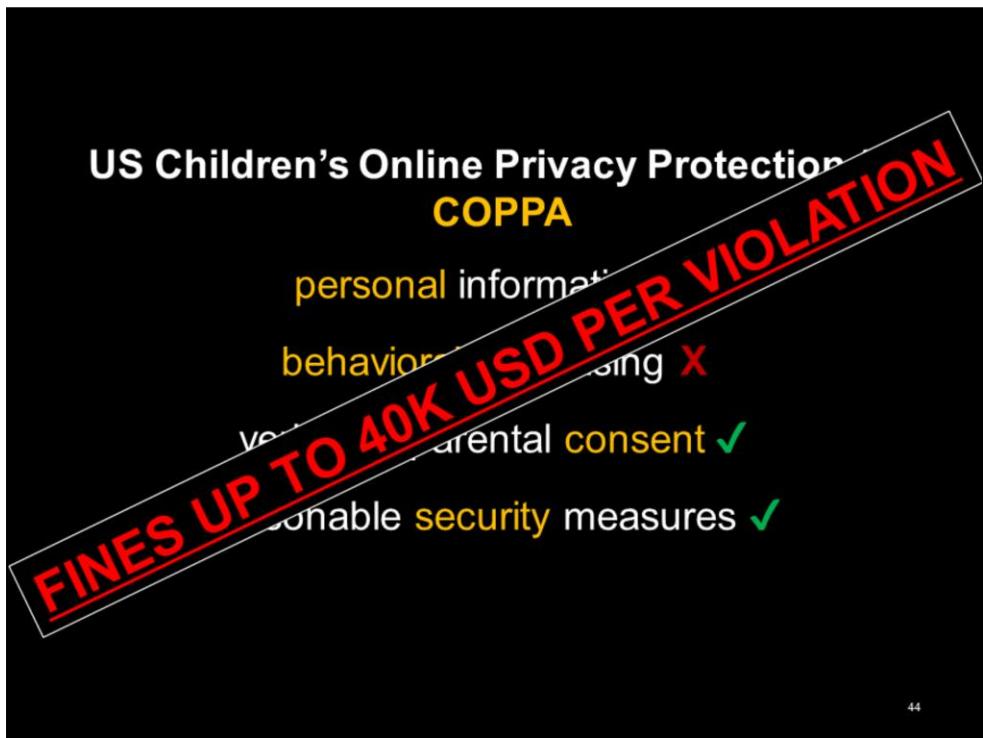
Not enough to just launch the app. Solution: explore with monkey. It's dumb!

Monkey did as well as undergrads 60% of the time in children's games. Results are a lower bound.

PERSONAL INFORMATION	PERSISTENT IDENTIFIERS
Owner Email Address	Hardware Serial Number
Phone Number	IMEI
GPS Latitude/Longitude	Wi-Fi MAC
Wi-Fi Router BSSID (MAC)	Android ID
Wi-Fi Router SSID (Name)	SIM Card ID
	Google Services Framework (GSF) ID
	Android Advertising ID (AAID)

43

Our system observes when apps access and share personal information, as well as unique persistent identifiers that can be used to track users over time and across services.

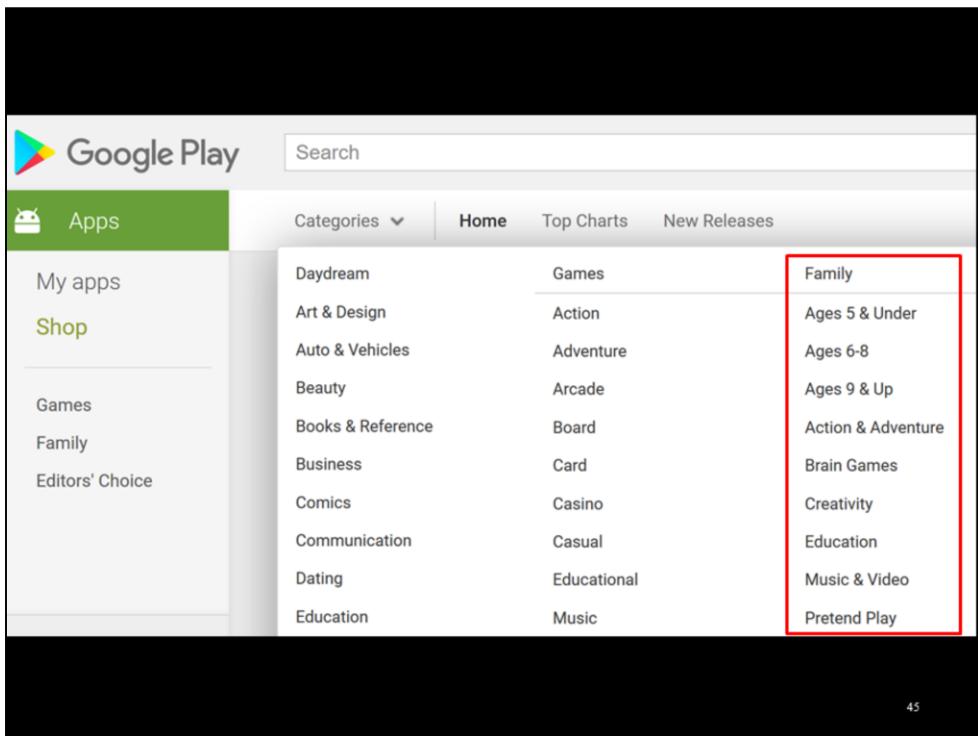


44

COPPA is one of the few comprehensive privacy laws in the US. It covers online services (like apps) that have users under 13 years of age.

Verifiable parental consent: Can take on the form of out-of-band methods like credit card verification or a phone call. Our system is fully automated with no direct human input, so observed data collection did not have consent.

Note that our analysis system is not specific to COPPA. It can be adapted to other regulatory measures such as GDPR and California's new online privacy law.



What apps does this law apply to? We looked at the “Family” category in the Google Play Store.

Designed for Families

Opt in to Designed for Families

Designed for Families is a developer program for apps and games designed specifically for children and family audiences. [Learn More](#)

Eligibility

All apps participating in the Designed for Families program must be relevant for children under the age of 13 and comply with the eligibility criteria below. App content must be appropriate for children. Google Play reserves the right to reject or remove any app determined to be inappropriate for the Designed for Families program.

7. You represent that apps submitted to Designed for Families are compliant with COPPA (Children's Online Privacy Protection Rule) and other relevant statutes, including any APIs that your app uses to provide the service.

<https://play.google.com/about/families/designated-for-families/program-requirements/>

46

Those are apps that have opted into the Designed for Families Program, or DFF for short.

DFF is opt-in. Participation is the dev saying kids are in the target audience. Google can reject or remove DFF apps not relevant to children.

DFF's requires devs to represent their apps **and bundled services** are COPPA compliant. For example, graphics, communications, analytics, and ads.



5,855 free “Designed for Families” apps

47

Apps collected between November 2016 and March 2018

Average 750K installs

Representing nearly 1900 developers

57% of “Designed for Families” apps are in potential violation

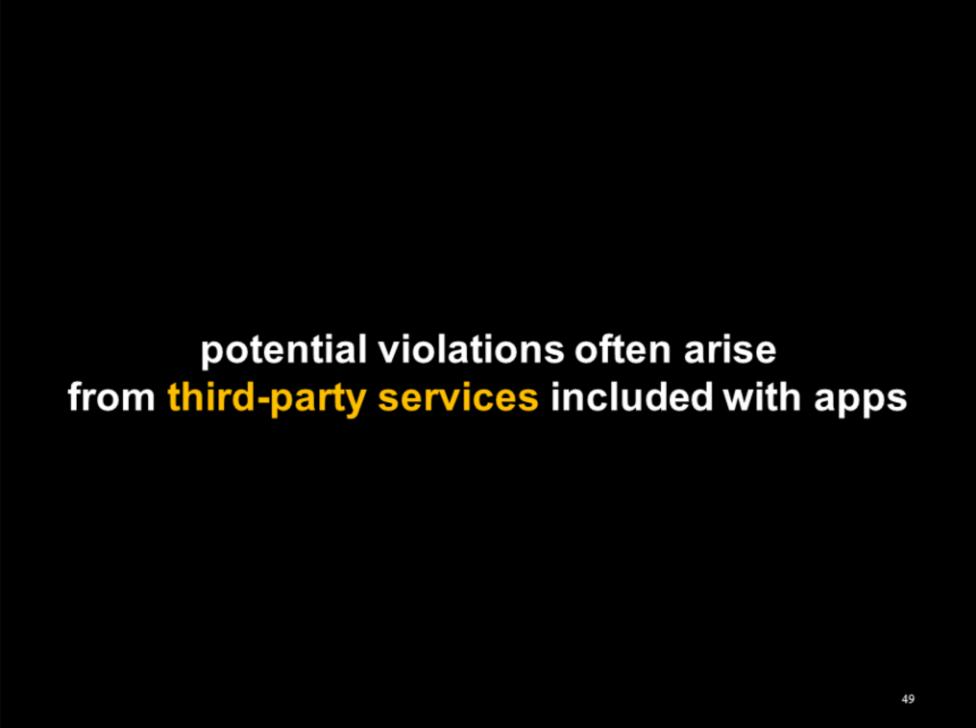
POTENTIAL VIOLATION	RATE (n=5,855)
▶  Personal information	4.8%
▶  Non-resettable identifiers	39%
▶  Potentially non-compliant services	19%
▶  Failure to take security measures	40%

48

The majority of our corpus was seen to be in potential violation of COPPA, in that they:

- Accessing and collecting email addresses, phone numbers, and fine geolocation
- Potentially enabling behavioral advertising through persistent identifiers
- Sharing user data and identifiers with SDKs that are themselves potentially non-compliant
- Not using standard security technologies

Note that some apps were observed engaging in more than one of these behaviors, so the percentages will add up to more than 57%.



**potential violations often arise
from third-party services included with apps**

49

We attributed most of these violations to various third-party services bundled with apps.

These services allow developers to expedite production by offering drop-in functionality, whether for graphics, communications, advertising, or analytics, among others.

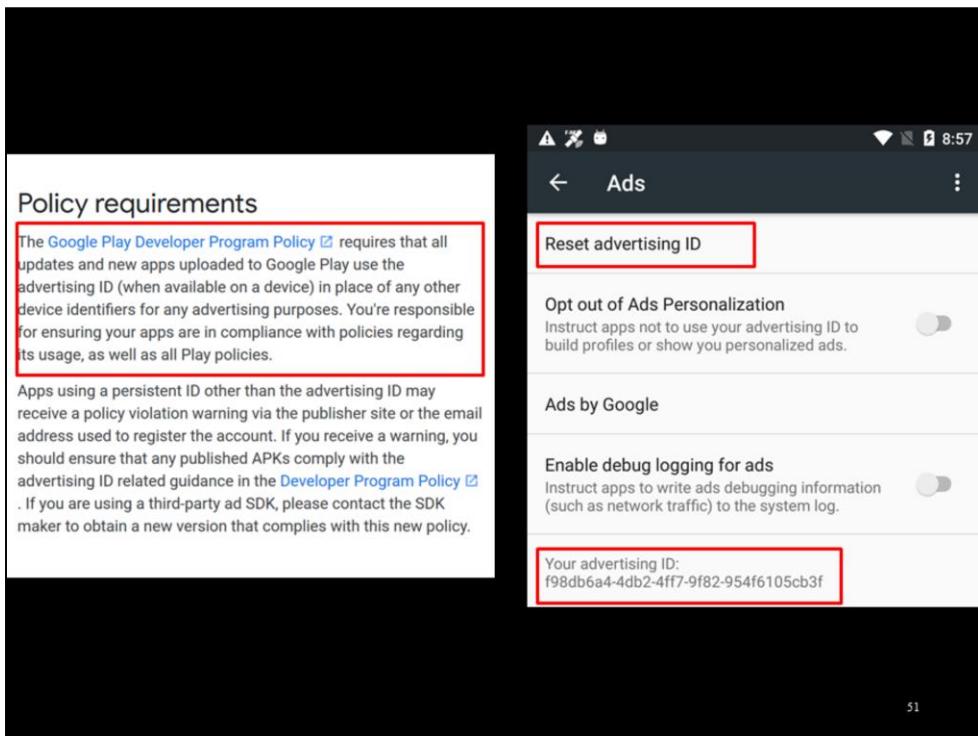


**potential violations persist
due to platform providers not enforcing terms**

50

We believe that these violations are prevalent because the gatekeepers in the mobile app space are not enforcing their own terms meant to protect end-users. (recall DFF requirements)

Google controls the Android operating system and the Play Store, which is the primary app distribution channel for Android. They are in an excellent position to conduct analysis similar to ours on all apps submitted to the Play Store, as well as secure the operating system to prevent potential abuses.



For example, COPPA prohibits behavioral advertising for children. Behavioral advertising uses persistent identifiers to build profiles of users by tracking individuals over time and across services.

Google has recognized the privacy implications of persistent identifiers, and in 2013 introduced the resettable Android Advertising ID (AAID) to give users (or parents) control over how advertisers track them. Since 2014, Google requires developers and advertisers to use this in lieu of non-resettable device identifiers like the IMEI and Wi-Fi MAC address.



**39% share the AAID along another identifier,
negating its privacy preserving benefits**

52

However, a large chunk of children's apps were seen sharing the AAID with another non-resettable identifier to the same destination, which defeats the purpose of the AAID. Although Google requires the use of the AAID, non-resettable identifiers remain available to apps.

AD PLATFORM	VIOLATION OF IDENTIFIER POLICY
Chartboost	> 99%
	> 99%
	98%
...	...
	3%
	2%
 DoubleClick <small>by Google</small>	1%

53

We found adherence to this AAID-only policy to vary among third-party ad networks. From nearly constant violation with Chartboost to nearly full compliance with Doubleclick (which is a Google company).

Full table in paper.



**19% share identifiers or personal information
with services not allowed in children's apps**

54

Not all third party services are appropriate for children, as claimed by those services themselves. We found nearly 1 in 5 DFF apps sharing personal information or identifiers with third-party services whose own terms of use prohibit their deployment in children's apps.

Recall that the apps we studied were opted into the Designed for Families program, indicating that the developers intended to include children in their apps' audience. Still, these same developers were found including these prohibited services.

not for children's apps



55

Presumably, these services prohibit their use in children's apps because these services may engage in non-COPPA-compliant data collection and processing.



Developer further agrees it will not integrate the Software into any Application or Beta Application (i) with end users who Developer has actual knowledge are under the age of 13, or (ii) that may be deemed to be a “Web site or online service directed to children” as defined under the Children’s Online Privacy Protection Act of 1998 (“COPPA”) and the regulations promulgated thereunder.

56

Crashlytics is a crash reporting service that allows developers to receive usage information about their apps in the wild. Crashlytics terms prohibit its use in children's apps.

Crashlytics

From Wikipedia, the free encyclopedia

Crashlytics is a Google-owned [Boston, Massachusetts](#)-based software company founded in May 2011 by entrepreneurs [Wayne Chang](#) and [Jeff Seibert](#).

57

Google owns Crashlytics, Android, and the Play Store. Google should be able to detect when its own service is integrated with children's apps, then take necessary steps to address that.



Potential COPPA violations are widespread, but the reality is regulatory agencies like the FTC have finite enforcement capability. COPPA, however, allows for industry self-regulation in the form of review and certification from designated safe harbor certifying bodies.

**industry self-regulation via safe harbors
has had no measurable positive effect**

59

However, we found that apps certified by safe harbors fared no better than DFF apps as a whole

POTENTIAL VIOLATION	DFF (n=5,855)	SAFE HARBOR (n=237)
 PERSONAL INFO	4.8%	10%
 NON-RESETTABLE IDENTIFIERS	39%	39%
 PROHIBITED SERVICES	19%	33%
 NO BASIC SECURITY MEASURES	40%	49%

60

In fact, they were in some cases worse.

There's a large body of economics research into adverse selection, in which bad actors are the ones most likely to participate in positive signaling activities.

We suspect safe harbors have had the unintended consequence of allowing potentially non-compliant apps to signal that they are indeed COPPA compliant.

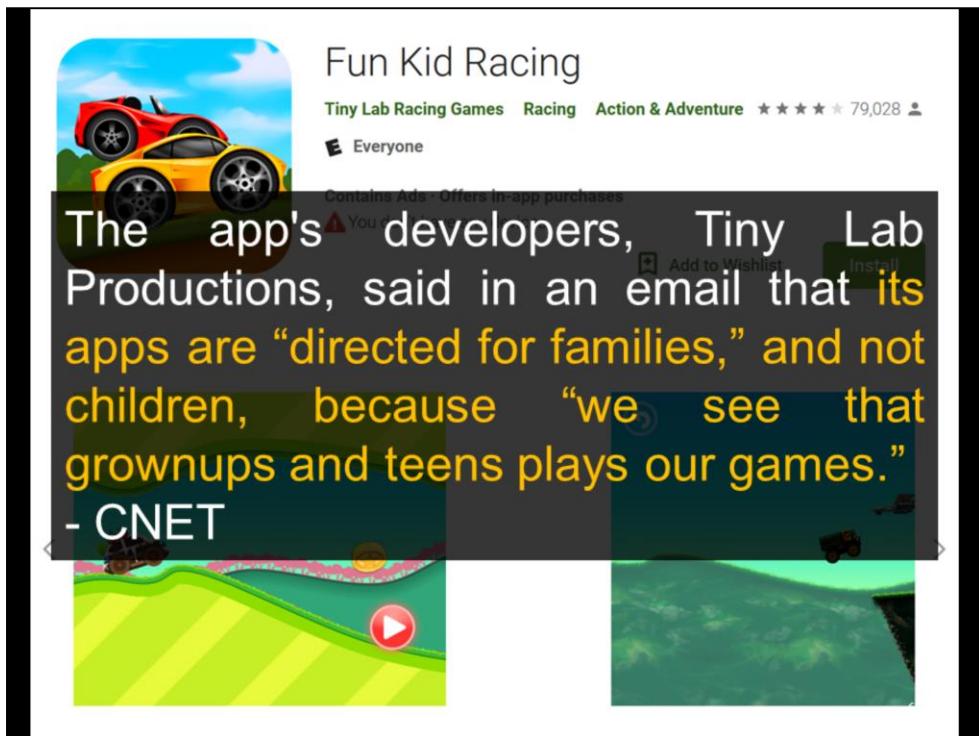


industry and regulators react

61

Our study has had an impact in industry and enforcement since its release last April.

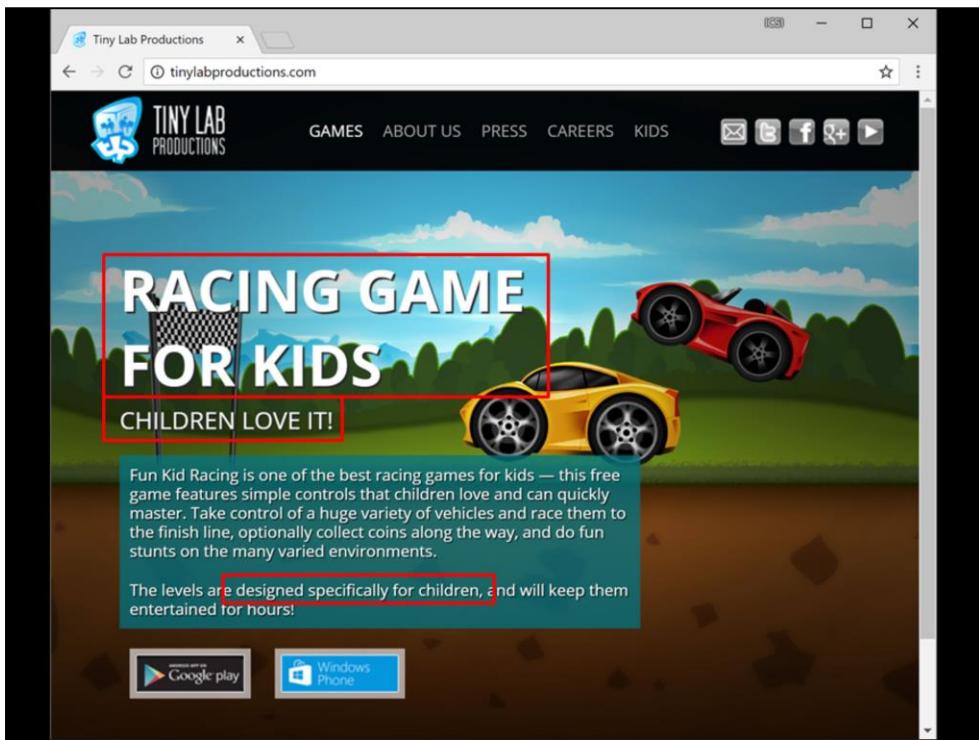
I'll close this presentation with an example of such impact.



In our study, we named Tiny Lab Productions's games as a popular example of the collection of personal information from children without verifiable consent.

Their game Fun Kid Racing has over 10M installs, and was seen collecting and sharing geolocation data with advertisers. Of Tiny Lab Production's 82 DFF games, we observed this behavior in 81 of them.

In response to our findings, Tiny Lab Productions stated to CNET that their games are not necessarily for children.



We have identified that 2,667 apps are potentially incorrectly listed as directed to “mixed audiences,” and “not primarily directed to children,” corresponding to ~51% of Designed for Families (DFF) apps from our original sample which are still listed on DFF.

Developers seem to have an incentive to miscategorize their apps as “not primarily directed to children” so they will be able to engage in defective “age gating,” thereby very likely causing children under 13 to enter ages over 13, allowing COPPA-prohibited behavioral advertising.

Email from our team to Google

64

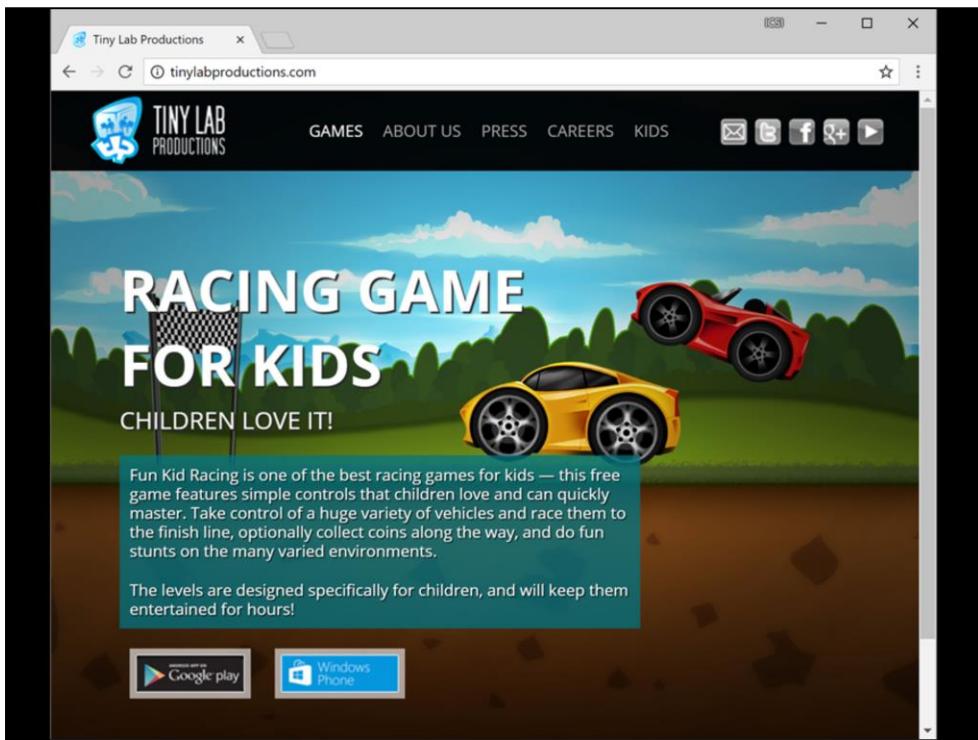
We reported Tiny Labs to Google, along with our results identifying all other DFF apps potentially violating COPPA and failing to meet Google’s own standards for DFF apps

115. In response, Google offered two primary rebuttals: (1) there was no mechanism to detect and prevent the issue “at scale”; and (2) more information was required on appropriate “heuristics” to support the conclusion that the Tiny Lab apps at issue were child directed.

65

Google responded to us saying that there was no way to detect these issues at scale, and that it was unclear that Tiny Labs was offering child-directed apps.

1) This was exactly the technology we developed and deployed in the course of this research



2) Definitely *not* for kids

The New York Times

How Game Apps That Captivate Kids Have Been Collecting Their Data

A lawsuit by New Mexico's attorney general accuses a popular app maker, as well as online ad businesses run by Google and Twitter, of violating children's privacy law.

By JENNIFER VALENTINO-DeVRIES, NATASHA SINGER, AARON KROLIK and MICHAEL H. KELLER SEPT. 12, 2018

<https://www.nytimes.com/interactive/2018/09/12/technology/kids-apps-data-privacy-google-twitter.html>

67

In September, the New Mexico Attorney General filed a suit, with Tiny Lab Productions and Google as co-defendants for violating children's privacy law.

A month later, Google appeared to reverse course: The company told Mr. Abromaitis it had identified a Tiny Lab app that should be designated for children. Google gave Tiny Lab a week to change that app and any others like it. Tiny Lab labeled 10 of its apps for children and used ad networks in them designed for children's apps. Google approved the updates but flagged more apps at the end of August, Mr. Abromaitis said, so he made another round of changes.

Then, this week, after inquiries from The Times, Google terminated Tiny Lab's account and removed all of its apps from the Play store, citing multiple policy violations.

<https://www.nytimes.com/interactive/2018/09/12/technology/kids-apps-data-privacy-google-twitter.html>

After facing scrutiny from the New York Times and the New Mexico AG's office, Google recently took a more aggressive stance towards Tiny Labs, taking down their apps after Tiny Labs failed to address the various privacy issues we identified in those products.

a bug that wasn't

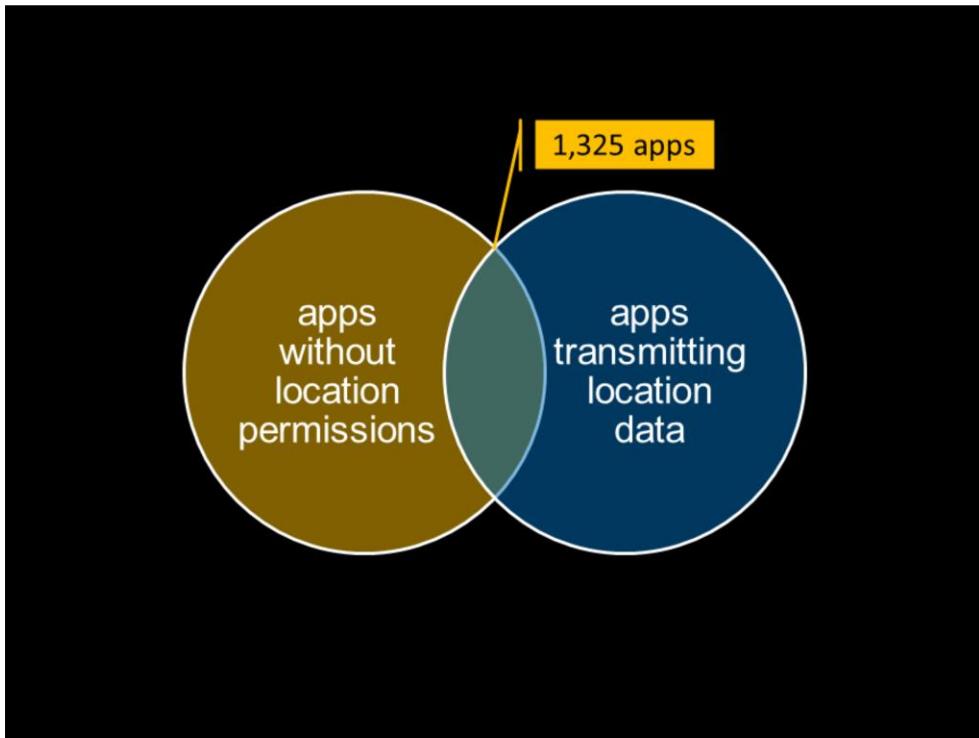
69

In the course of developing and refining this app testing infrastructure, we encountered a “critical bug” that turned out to be something more interesting



One day as a sanity check, I asked our database of app behaviors, “give me all that apps that sent location data but never declared permissions to access the phone’s location.”

This intersection should be null.



Instead, the database turned up over 1300 apps that match this criteria.

I panicked for a second because

Despite the failures of permission systems
they serve an **important purpose**.
At the very least, if an app is **denied**
permission, it **must not** access
resources protected by the permission.

From Reardon's talk: apps that don't hold appropriate permissions shouldn't be able to access those resources



73

The Android permissions system can be circumvented, often through the permissions system itself

50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System

Joel Reardon
University of Calgary
AppCensus, Inc.

Amit Elazari Bar On
U.C. Berkeley

Álvaro Feal
IMDEA Networks Institute
Universidad Carlos III de Madrid

Narseo Vallina-Rodríguez
IMDEA Networks Institute / ICSI
AppCensus, Inc.

Primal Wijesekera
U.C. Berkeley / ICSI

Serge Egelman
U.C. Berkeley / ICSI
AppCensus, Inc.

Abstract

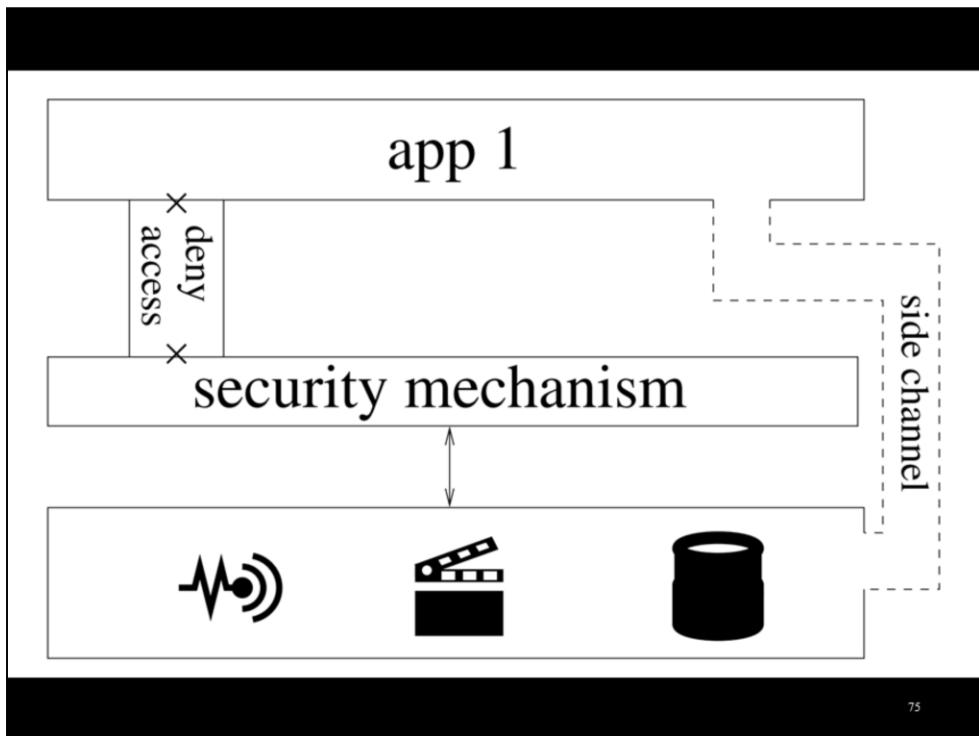
Modern smartphone platforms implement permission-based models to protect access to sensitive data and system resources. However, apps can circumvent the permission model and gain access to protected data without user consent by using both covert and side channels. Side channels present in the implementation of the permission system allow apps to access protected data and system resources without permission; whereas covert channels enable communication between two colluding apps so that one app can share its permission-protected data with another app lacking those permissions. Both pose threats to user privacy.

is crucial to protect this information from unauthorized access. Android, the most-popular mobile phone operating system [75], implements a permission-based system to regulate access to these sensitive resources by third-party applications. In this model, app developers must explicitly request *permission* to access sensitive resources in their Android Manifest file [5]. This model is supposed to give users control in deciding which apps can access which resources and information; in practice it does not address the issue completely [30, 86].

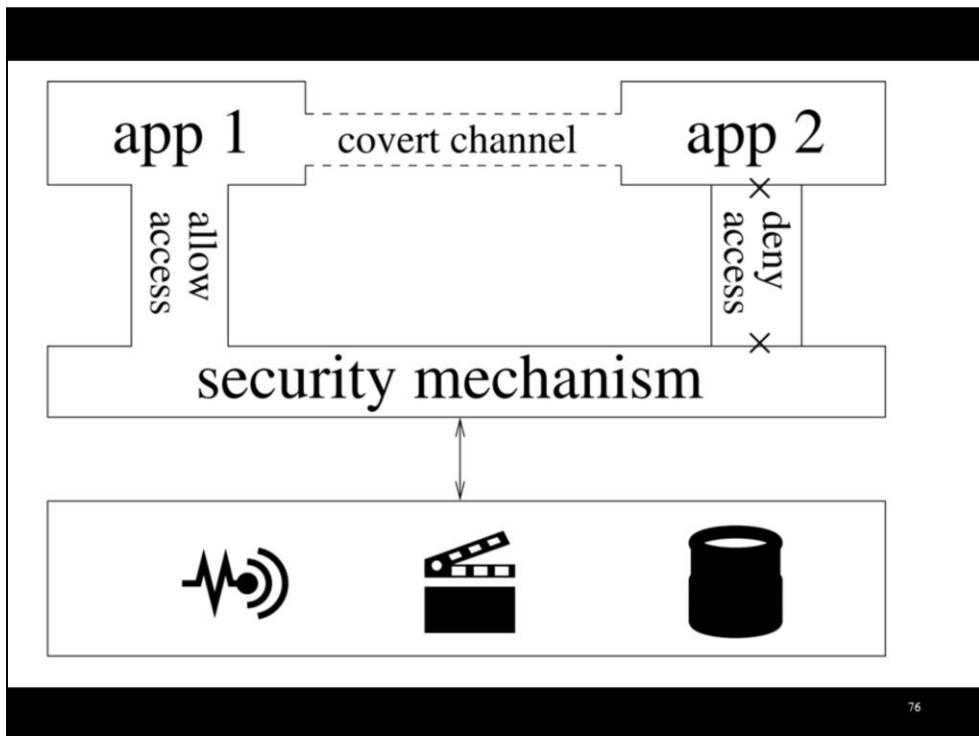
The Android operating system sandboxes user-space apps to prevent them from interacting arbitrarily with other running apps. Android implements isolation by assigning each

74

This “bug” resulted in a USENIX paper



Example side channels: EXIF data; /proc/net



Example covert channel: App 1 holds appropriate permissions, writes sensitive data to shared storage, App 2 doesn't have permissions but can read from storage

THANKS!



<https://irwinreyes.com>



@irwinreyescom



irwin.reyes@twosixlabs.com



Berkeley
UNIVERSITY OF CALIFORNIA

ioreyes@icsi.berkeley.edu