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

Joel Reardon
University of Calgary
AppCensus, Inc.

Álvaro Feal
IMDEA Networks Institute
Universidad Carlos III de Madrid

Primal Wijesekera
U.C. Berkeley / ICSI

Amit Elazari Bar On
U.C. Berkeley

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

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

Smartphone are general purpose computers that store a great deal of personal sensitive information. Apps are prevented from accessing this information at will through the use of a *permission system* at the operating-system level. These security mechanisms are reasonable because we carry our smartphones alongside us all day, and they can gain access to our intimate communications and social network, our web browsing history, our location at all times—even if the GPS is disabled. When apps are denied permissions, however, they still have options to cheat the permission system by using side and covert channels. In our research we found a small number of such channels being actively exploited when we tested Google Play Store apps.

Are mobile permission models bullet-proof?

There are lots of valid criticisms for the current permissions system. Users cannot reliably understand what permissions mean or why they are needed. Apps request more permissions than necessary. Users don't have easy means to find alternate apps that request fewer permissions, or to omit search results for apps that request dangerous permissions, like being able to turn on your microphone at all times.

The increasing presence of third-party SDKs in mobile applications amplifies

the dissemination of personal data from mobile applications to online services. Most developers use third-party SDKs in their apps for purposes like advertising, analytics, crash reporting, or social network integration [5]. Both Android and iOS permission models allow third-party SDKs to piggyback on the permissions that the user grants to the host app. Unfortunately, users cannot distinguish when a permission is given to enable a feature in the app or if it is going to be used by a data-hungry third-party SDK [5].

StartApp's official guidance for integrating its SDK into apps give a perfect example of this problem. It tells developers it improves performance if they add extra permissions for location, Bluetooth, and silent starting on boot [4], that is, it tells developers to add location access to their apps even though the apps that would add it have no legitimate need for it. Users aren't made aware that permissions been requested by an advertising library that simply wants to track them and harvest their private information.

Additionally, mobile apps can circumvent the permission model and gain access to protected data without user consent by using both *covert and side channels*, attacks described in Figure 1. Side channels manifest through vulnerabilities present in the implementation of the OS permission system which allow apps to access protected data and system resources without permission. Covert channels manifest when inter-app communication, which may be legitimate, is leveraged for illegitimate purposes, such as having one app abuse its privileges by acting as a facade for another app's desire to access permission protected data.

Discovering covert and side channels in the wild

Previous research focused on understanding personal data collection using the system-supported access control mechanisms (*i.e.*, Android permissions). The research community has also explored the potential for covert channels in Android using local sockets, shared storage [2], and other unorthodox means, such as using vibrations to send data and accelerometers to receive it [1]. Accelerometer data can further act as a side channel to uniquely identify the user [9, 11] or infer demographic data such as their gender [3]. However, there has been little research in detecting and measuring at scale the prevalence of both covert and side channels in apps that are available in the Google Play Store.

Instead of imagining new channels, our USENIX Security 2019 paper *"50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System"* focuses on collecting evidence of apps abusing side and covert channels in practice [7]. We leveraged our AppCensus app auditing platform to search for instances of Android applications disseminating permission-protected data over the network without requesting the permission to access it. We then reverse engineer the apps and third-party libraries responsible for this behavior to determine how the unauthorized access occurred.

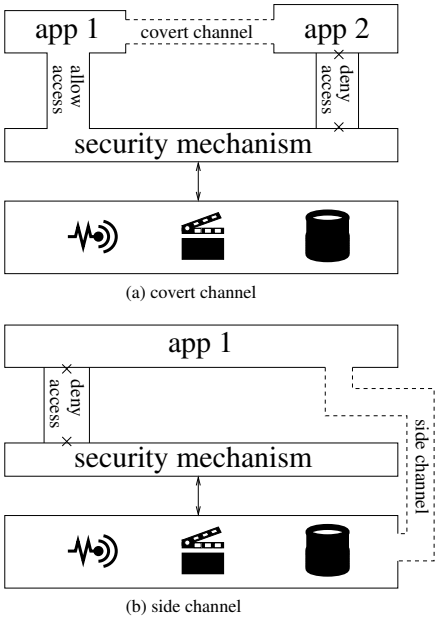

Figure 1: Covert and side channels. (a) A security mechanism allows `app1` access to resources but denies `app2` access; this is circumvented by `app2` using `app1` as a facade to obtain access over a communication channel not monitored by the security mechanism. (b) A security mechanism denies `app1` access to resources; this is circumvented by accessing the resources through a side channel that bypasses the security mechanism.

It is important to note that AppCensus is not a regular security-oriented sandbox: detecting and analyzing both privacy abuses and regulatory violations require specific research methods. To that end, AppCensus implements mechanisms to exhaustively monitor apps' runtime behavior and personal data leaks at the system and network level, including a TLS man-in-the-middle proxy. Then, we leverage heuristics inspired by different regulatory frameworks to contextualize these observations and to hunt potential abuses and violations.

Research findings

We automatically executed over 88,000 Android apps in our AppCensus platform to see when permission-protected data was transmitted by the device, and scanned the permissions that apps requested to see which ones *should not* be *even able* to access the transmitted data in the first place. We focus on a subset of the *dangerous* permissions that prevent apps from accessing location data and unique identifiers. We grouped our findings by *where* on the Internet *what* data type was sent, as this allows us to attribute the observations to the actual app developer or embedded third-party libraries. We then reverse engineered

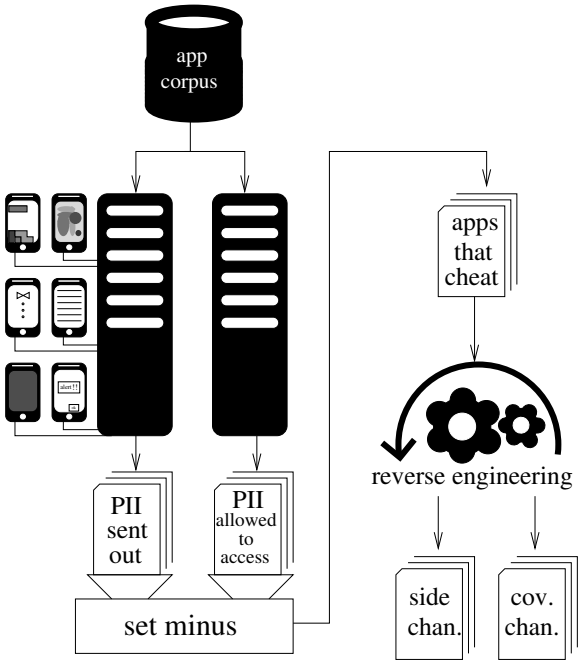

Figure 2: Overview of our analysis pipeline. Apps are automatically run and the transmissions of sensitive data are compared to what would be allowed. Those suspected of using a side or covert channel are manually reverse engineered.

the responsible component to determine exactly how the data was accessed so that we could statically analyze our entire dataset to measure the prevalence of each attack. We found the following side and covert channels being exploited in Google Play Store apps:

- We discovered apps getting the BSSID of the connected WiFi Access Point (i.e., the router's MAC address) by reading the OS ARP cache (`/proc/net/arp`), which was not protected by permissions. This information can be used as a surrogate for location data. We found 5 apps exploiting this vulnerability and 355 with the pertinent code to do so.
- We discovered Unity (a popular third-party cross-platform game engine and advertising network) obtaining the device MAC address of the device using `ioctl` system calls. This information can be used to track users even if they factory reset their devices. We found 42 apps exploiting this vulnerability and 12,408 apps with the pertinent code to do so. We realized (after our paper was published) that starting from the version of Android we used (Marshmallow) all attempts to access the MAC address of the device returns a fake value of `02:00:00:00:00:00`—even if the ACCESS_NETWORK_STATE permission is granted: therefore all 711 apps that transmitted the MAC address must have accessed it this way.

- We also discovered that third-party libraries provided by two Chinese companies—Baidu and Salmonads—independently make use of the SD card as a covert channel, so that when an app can read the phone's IMEI, it stores it for other apps that cannot. We found 159 apps with the potential to exploit this covert channel and empirically found 13 apps doing so.
- We found one app, Shutterfly, that used picture metadata as a side channel to access precise historical location information despite not holding location permissions. It included code that *processed* location from the raw EXIF data; that is, it copied the data intentionally instead of simply uploading photos and having location data by mistake.

The impact of our work

The permissions system is not perfect, but it serves an important purpose. Requesting permission serves as a system to give users *notice* about the app's behavior; users installing apps serves further as a system of *consent*. The use of deceptive practices like covert and side channels is unacceptable as they not only undermine users' privacy and consumer rights, but they also give rise to legal and regulatory concerns. Circumventing the permissions system means that neither notice was given nor consent obtained. In one case, the third-party library OpenX first *tries* to obtain the WiFi BSSID first through the permission system, and only goes the cheating route through the ARP cache if it sees that it was denied access.

Data protection legislation around the world, like the General Data Protection Regulation (GDPR) in Europe or the California Consumer Privacy Act (CCPA), enforce transparency on the data collection, processing, and sharing practices of mobile applications. In this regulatory context, designing and using these techniques suggests an actual attempt to access data without user consent. Developers and SDK providers implementing these techniques have to take extra measures to set up covert channels or discover side channels that can be exploited. We responsibly disclosed our findings to Google so they may address the issues in the Android operating system as well as the U.S. Federal Trade Commission (FTC). Google has given us a bug bounty for our efforts.

Our Stepping Stones

This work originates from a line of work designed at improving the accuracy and usability of the Android permission system [10]. Anyone who has installed an app on Android and paid attention to the permissions that are requested has probably run into one that demands permissions that fall well outside their scope, like an alarm clock app that needs to read your SMSs. (The best expla-

nation we've come up with is that it allows someone trusted to set important alarms for you after you've gone to sleep—like if there's going to be a huge dump of fresh snow in the mountains and they'll come to pick you up.)

Part of this earlier work involved instrumenting the permission system to track permission usage by apps and collecting ground truth data about how users would prefer to handle those permission requests. This knowledge was used to inform a machine learning classifier that significantly improved the permission granting accuracy over the existing ask-on-first-use (and was much better than the ask-on-install ultimatum).

This work was followed by a field study where we built a modified version of Android that actually enforced denying permissions. We did this gracefully when possible and used both user input and our machine learning classifier. Users liked the control they got and our results from earlier studies were validated. One observation from our field studies was that apps made frequent requests to access data protected by sensitive permissions.

In parallel, another line of research was studying trackers in the mobile ecosystem and personal data dissemination over the network [6]: all the data-hungry ads and analytics companies that are spying on users. This study took advantage of a purpose-built man-in-the-middle VPN on Android, the Lumen Privacy Monitor, a tool that can monitor applications' traffic locally on the device, even if encrypted. Lumen allowed us build a database of all network traffic going to different organizations that an app contacts.

These lines of research joined together when some of us decided to read the Children's Online Privacy Protection Act (COPPA)—a particularly strong privacy regulation with serious consequences for violations—and realize that based on what we've seen in practice, there's *no way* that *all* of these apps are in compliance. Plus, we have have all the tools to monitor for this. We combined our OS instrumentation with our traffic monitoring to obtain evidence of applications' actual runtime behavior regarding *when* personal data is accessed and *where* it is sent. We could automate our analysis and thus scale our study by simulating human interaction with apps using the Android Automator Monkey, which is essentially a UI fuzzer for testing purposes.

Our findings about COPPA compliance in children-oriented Android apps were shocking [8]. The majority of children's games are sending persistent identifiers to ads and analytics companies capable of tracking them. Ten percent are sending the IMEI of the device, which is like an unresetable super cookie of infinite tracking. *Four percent* were sending *precise geolocation*, for which COPPA requires *verified* parental consent to access. How on earth can a company feel confident in having verified parental consent from a system that randomly clicks and swipes, we'll never know!

When we looked to see at the ones that we know *used* the location permission while running but we didn't catch sending location, we suddenly found a bunch

of obfuscated location sending happening. This includes the company StartApp, which Google lists as one of their accepted children advertisers in their updated designed for families program.[1] They were using a Vernam-style cipher to XOR in two repeating masks (they were `$T@RT@PP` and `ENCRYPTIONKEY`) and in doing so were transmitting precise geolocation and even WiFi scan data including router MAC and signal strength.

From all these stepping stones we end up at this work. We have the ability to run lots of apps at scale, to monitor their network traffic, and to scrutinize the permissions that they request in runtime. So we compared these two sets: what's the data an app is *allowed* to access, and what's the data that an app *actually* sends out on the Internet. Are there any transmissions by an app that didn't have permission to access it in the first place?

Our Confession

Now it's time for our confession. Our original goal in our methodology was not to discover and disclose these side and covert channels. We actually discovered these attacks by chance: we were actually looking for bugs in our own code. That is, we implicitly assumed that the Android permission system is absolutely sound, and we were looking for false positives in our dataset because, as we imagined, if we flagged the transmission of the IMEI without the READ_PHONE_STATE permission, it *must* be a bug with our code.

A few false positives and negatives can be expected with such large-scale work, and we spot checked lots of transmissions of PII that we flagged but by no means manually every transmission (so we'll have some false positives). And we looked at lots of packets trying to find all sorts of obfuscations but there's many that still confound us (so we expect some false negatives as well). Still, as long as we do enough manual checking of our findings, the false positive rates are statistically low enough to not have any impact on *headline results* like four percent of apps sending location.

But our study on rampant (potential) privacy violations in thousands of children's games was getting media and regulatory attention. This prompted us to become extra certain of our findings. Being confident about the average value is no longer enough and rooting out any false positives became even more crucial. We can live with the false negatives (where we don't catch a company who is actually sending data) but now false positives have become critical to avoid, because even one casts doubt on any *specific* finding that we claim. We, for example, got a letter from one of the lawyers at IronSource who did not like our characterization of their behaviors. (Their letter resulted in us then double checking our results for IronSource, verifying our initial findings and actually finding *more* things we had missed!)

[1] `https://android-developers.googleblog.com/2019/05/building-safer-google-play-for-kids.html`

So we went looking for false positives. We took all the data that we saw sent and filtered out the ones where the app had the corresponding permission assuming that what is left must all be false positives. And in fact we did find some! One favorite was the fact that we did our tests in Berkeley, California, which has an area code of 510—it just happened that the UNIX timestamp began with 1510 during which time we did some of our tests and so there's a block of time where a harmless timestamp is misconstrued as apps transmitting the user's phone number.

Another was the fact that IP-based geolocation happened to be surprising accurate for IPs from our research institute. Perhaps this was because we uploaded both our IP and location thousands of times after running all these apps and eventually the Internet learned where this IP was. Digging deeper, however, we found that this did not replicate at other locations and with other IPs. Finally, some apps sent really invasive fingerprints which including the hostname of our own machines that built our custom Android version, and it just so happened that the SSID of our WiFi router was a substring of that.

Our hunt was a useful exercise and we fixed all the false positives that we found, making our tools more robust and reliable. But we also found true positives. We found actual transmissions of data, that was the correct values and (unlike incoming geolocation) was first seen as an outgoing transmission from the app. It turns out that we found evidence consistent with the use of side and covert channels, and in order to figure out what exactly is going on we had start doing some reverse engineering. The results of this exercise were those five side and covert channels we presented earlier in the article: `ioctl`s, EXIF metadata, ARP cache, and plain old sharing data on the SD card. And in so doing we put app and SDK developers on notice that we are looking for these kinds of deceptive and fraudulent practices going forwards.

Acknowledgments

This work was supported by the U.S. National Security Agency's Science of Security program (contract H98230-18-D-0006), the Department of Homeland Security (contract FA8750-18-2-0096), the National Science Foundation (grants CNS-1817248 and grant CNS-1564329), the Rose Foundation, the European Union's Horizon 2020 Innovation Action program (grant Agreement No. 786741, SMOOTH Project), the Data Transparency Lab, and the Center for Long-Term Cybersecurity at U.C. Berkeley. The authors would like to thank John Aycock, Irwin Reyes, Greg Hagen, René Mayrhofer, Giles Hogben, and Refjohürs Lykkewe.

References

[1] A. Al-Haiqi, M. Ismail, and R. Nordin. A new sensors-based covert channel on android. *The Scientific World Journal*, 2014, 2014.

[2] C. Marforio, H. Ritzdorf, A. Francillon, and S. Capkun. Analysis of the communication between colluding applications on modern smartphones. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 51–60. ACM, 2012.

[3] Y. Michalevsky, D. Boneh, and G. Nakibly. Gyrophone: Recognizing speech from gyroscope signals. In *USENIX Security Symposium*, pages 1053–1067, 2014.

[4] Shelly Milo. Startapp sdk android—android (standard). `https://support.startapp.com/hc/en-us/articles/360002411114-Android-Standard-`, 2019. Accessed: Sept 8th, 2019.

[5] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich, and P. Gill. Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2018.

[6] A. Razaghpanah, N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, P. Gill, M. Allman, and V. Paxson. Haystack: In Situ Mobile Traffic Analysis in User Space. *arXiv preprint arXiv:1510.01419*, 2015.

[7] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 603–620, Santa Clara, CA, August 2019. USENIX Association.

[8] I. Reyes, P. Wijesekera, J. Reardon, A. Elazari Bar On, A. Razaghpanah, N. Vallina-Rodriguez, and S. Egelman. "Won't Somebody Think of the Children?" Examining COPPA Compliance at Scale. *Proceedings on Privacy Enhancing Technologies*, 2018(3):63–83, 2018.

[9] L. Simon, W. Xu, and R. Anderson. Don't interrupt me while i type: Inferring text entered through gesture typing on android keyboards. *Proceedings on Privacy Enhancing Technologies*, 2016(3):136–154, 2016.

[10] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov. Android permissions remystified: A field study on contextual integrity. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 499–514, Washington, D.C., August 2015. USENIX Association.

[11] J. Zhang, A. R. Beresford, and I. Sheret. Sensorid: Sensor calibration fingerprinting for smartphones. In *Proceedings of the 40th IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2019.