50 Ways to Leak Your Data

An Exploration of Apps' Circumvention of the Android Permissions System

JOEL REARDON, ÁLVARO FEAL, PRIMAL WIJESEKERA, AMIT ELAZARI BAR ON, NARSEO VALLINA-RODRIGUEZ, AND SERGE EGELMAN

Joel Reardon is an Assistant Professor at the University of Calgary. He received his master's degree from the University of Waterloo, his doctoral degree from the ETH Zurich, and spent a postdoctoral year at UC Berkeley and the International Computer Science Institute (ICSI). His research interests relate to security and privacy, including storage compliance issues as well as systems to make it easier to use. He also loves mountains, bicycles, and writing poetry. joel@moosematch.com

Álvaro Feal is a second-year PhD student working at IMDEA Networks Institute. He focuses on analyzing privacy threats to web and mobile applications from a technical and regulatory perspective. Prior to working at IMDEA Networks, Alvaro interned at IMDEA Software, working in Android privacy and anonymous communication systems. He has published in peer-reviewed conferences such as ACM IMC, USENIX Security, and workshops like IEEE Consumer Protection (ConPro). Álvaro's work received a Distinguished Paper Award at USENIX Security 2019. alvaro.feal@imdea.org

Primal Wijesekera is a Research Scientist in the Usable Security and Privacy Group at the International Computer Science Institute at Berkeley and is a Postdoctoral Researcher in the Electrical Engineering and Computer Science Department at UC Berkeley. His prior work includes contextual permission models for Android, mobile app analysis for privacy and security violations. His recent work focuses on smart speakers for the home, vulnerability discovery mechanisms in the wild, and ecosystems surrounding fake news. primal@berkeley.edu

Smartphones are general-purpose computers that store a great deal of sensitive personal 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 software development kits (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 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 between a permission given to enable a feature in the app and one to be used by a data-hungry third-party SDK [5].

StartApp's official guidance for integrating its SDK into apps provides a perfect example of this problem. It tells developers that it will improve 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 would have no legitimate need for such access. Users aren't made aware when permissions have 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 that 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.

Amit Elazari Bar On is a Director of Global Cybersecurity Policy at Intel Corporation and a Lecturer at UC Berkeley's School of Information. She holds a JSD from UC Berkeley School of Law and graduated summa cum laude in gaining three prior degrees. Her research in cybersecurity law and policy has appeared in leading journals, has been presented at conferences such as RSA, Black Hat, and USENIX Security, and was featured at leading news sites such as the *New York Times*. She practiced law in Israel. amit.elazari@berkeley.edu

Narseo Vallina-Rodriguez is an Assistant Research Professor at IMDEA Networks and a Research Scientist at ICSI in Berkeley. His research interests fall in the area of network measurements, privacy, and security. Narseo's work has received distinguished paper awards at USENIX Security 2019, ACM IMC 2018, and ACM CoNEXT 2014, and was awarded the IETF Applied Networking Research Prize in 2016. Narseo's research in mobile privacy has influenced industry practices and regulation and has been covered by international media outlets. narseo.vallina@imdea.org

Serge Egelman is the Research Director of the Usable Security and Privacy Group at the International Computer Science Institute. He conducts research to help people make more informed online privacy and security decisions. He has received the USENIX Distinguished Paper Award, seven ACM CHI Honorable Mention Awards, and the SOUPS Impact and best paper awards; his research has been cited in numerous lawsuits and regulatory actions, as well as being featured in the *New York Times, Washington Post,* and *Wall Street Journal.* egelman@cs.berkeley

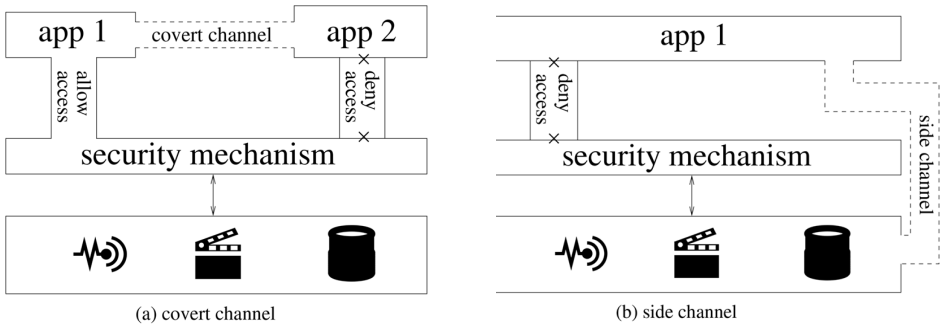

Figure 1: Covert and side channels. (a) A security mechanism allows app1 access to resources but denies app2 access; this is circumvented by app2 using app1 as a facade to obtain access over a communication channel not monitored by the security mechanism. (b) A security mechanism denies app1 access to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

Discovering Covert and Side Channels in the Wild

Previous research focused on understanding personal data collection using 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 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 '19 paper 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 engineered the apps and third-party libraries responsible for this behavior to determine how the unauthorized access occurred.

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 system and network levels, including a TLS man-in-the-middle proxy. Then, we leverage heuristics inspired by different regulatory frameworks to contextualize these observations and to hunt for 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 even be able* to access the transmitted data in the first place (Figure 2). We focused 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 data was sent and *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 the responsible component to determine exactly how the data was accessed so that we could statically analyze our entire data set to measure the prevalence of each attack. We found the following side and covert channels being exploited in Google Play Store apps:

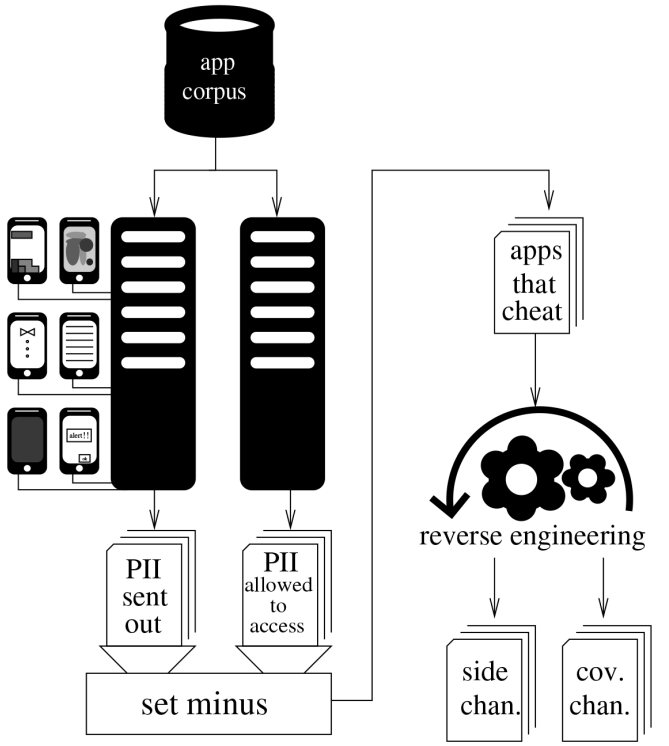

Figure 2: Overview of our analysis pipeline. Apps are automatically run, and the transmissions of sensitive data are compared to what would be allowed. Those suspected of using a side or covert channel are manually reverse engineered.

- 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 five 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 return 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 further serves 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 notice was not given nor consent obtained. In one case, the third-party library OpenX first *tried* to obtain the WiFi BSSID through the permission system and only went the cheating route through the ARP cache when it saw 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 could address the issues in the Android operating system, as well as the US Federal Trade Commission (FTC). Google has given us a bug bounty for our efforts.

Our Stepping Stones

Our research originates from a line of work designed to improve 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 SMSes. The best explanation we've come up with is that this 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 involved studying trackers—all the data-hungry ads and analytics companies that are spying on users—in the mobile ecosystem and personal data dissemination over the network [6]. 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 to build a database of all network traffic going to different organizations that an app contacts.

Spying on Children

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 realized that, based on what we've seen in practice, there's *no way* that *all* of these apps are in compliance. Plus, we 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 un-resettable super cookie of infinite tracking. *Four percent* were sending *precise geolocation*, for which COPPA requires *verified* parental consent to access. How on earth a company can feel confident in having verified parental consent from a system that randomly clicks and swipes, we'll never know!

For apps that we know *used* the location permission while running but that we didn't catch sending location, we found a bunch of obfuscated location sending happening. This category of app includes the company StartApp, which Google lists as one of their accepted children advertisers in their updated designed for families program (https://android-developers.googleblog.com/2019/05/building-safer-google-play-for-kids.html). StartApp was using a Vernam-style cipher to XOR in two repeating masks (`$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 were actually looking for bugs in our own code but discovered these attacks by chance. That is, we implicitly assumed that the Android permission system was absolutely sound and were looking for false positives in our data set because, as we imagined, if we flagged the transmission of the IMEI without the `READ_PHONE_STATE` permission, it *must* be the result of a bug in our code.

A few false positives and negatives can be expected with such large-scale work, and we spot-checked lots of flagged transmissions of PII 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 are 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 false positive casts doubt on any *specific* finding that we claim. For example, in response to a letter from one of the lawyers at IronSource who did not like our characterization of their behaviors, we double-checked our results in order to verify our initial findings and actually found more things we had missed!

So we went looking for false positives. We filtered out all the data where the app had the corresponding permission, assuming that what was 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 so happened that some of our testing began with the UNIX timestamp 1510 and so there's a block of time during which a harmless timestamp was misconstrued as apps transmitting the user's phone number.

Another was the fact that IP-based geolocation happened to be surprisingly 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, 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 carrying the correct values and (unlike incoming geolocation) 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 was going on we had to start reverse engineering. The results of this exercise were those four side and covert channels we presented earlier in the article: ioctls, 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, going forward, we are looking for these kinds of deceptive and fraudulent practices.

Acknowledgments

This work was supported by the US 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 UC 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*, September 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* (ACM, 2012), pp. 51–60.

[3] Y. Michalevsky, D. Boneh, and G. Nakibly, "Gyrophone: Recognizing Speech from Gyroscope Signals," in *Proceedings of the 23rd USENIX Security Symposium (USENIX Security '18)*, pp. 1053–1067: https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-michalevsky.pdf.

[4] S. Milo, StartApp SDK Android—Android (Standard): https://support.startapp.com/hc/en-us/articles/360002411114-Android-Standard-, 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 '18)*.

[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:1510.01419v1*, (2015), pp. 1–13.

[7] J. Reardon, Á. Feal, P. Wijesekera, A. Elazari Bar On, N. Vallina-Rodriguez, and S. Egelman, "50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System," in *Proceedings of the 28th USENIX Security Symposium (USENIX Security '19)*, pp. 603–620: https://www.usenix.org/conference/usenixsecurity19/presentation/reardon.

[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," in *Proceedings on Privacy Enhancing Technologies*, 2018, no. 3, pp. 63–83.

[9] L. Simon, W. Xu, and R. Anderson, "Don't Interrupt Me While I Type: Inferring Text Entered through Gesture Typing on Android Keyboards," in *Proceedings on Privacy Enhancing Technologies*, 2016, no. 3, pp. 136–154.

[10] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov, "Android Permissions Remystified: A Field Study on Contextual Integrity," in *Proceedings of the 24th USENIX Security Symposium (USENIX Security '15)*, pp. 499–514: https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-wijesekera.pdf.

[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.