Bug Fixes, Improvements, ... and Privacy Leaks

A Longitudinal Study of PII Leaks Across Android App Versions

Jingjing Ren*, Martina Lindorfer†, Daniel J. Dubois*,
Ashwin Rao‡, David Choffnes* and Narseo Vallina-Rodriguez§
*Northeastern University †UC Santa Barbara ‡University of Helsinki §IMDEA Networks Institute and ICSI

***Abstract*—Is mobile privacy getting better or worse over time? In this paper, we address this question by studying privacy leaks from historical and current versions of 512 popular Android apps, covering 7,665 app releases over 8 years of app version history. Through automated and scripted interaction with apps and analysis of the network traffic they generate on real mobile devices, we identify how privacy changes over time for individual apps and in aggregate. We find several trends that include increased collection of personally identifiable information (PII) across app versions, slow adoption of HTTPS to secure the information sent to other parties, and a large number of third parties being able to link user activity and locations across apps. Interestingly, while privacy is getting worse in aggregate, we find that the privacy risk of individual apps varies greatly over time, and a substantial fraction of apps see little change or even improvement in privacy. Given these trends, we propose metrics for quantifying privacy risk and for providing this risk assessment proactively to help users balance the risks and benefits of installing new versions of apps.**

I. Introduction

As mobile devices and apps become increasingly present in our everyday lives, the potential for accessing and sharing personal information has grown. The corresponding privacy risks from using these apps have received significant attention, not only from users who are at risk [55], but also from regulators who enforce laws that protect them [26].

A key problem with the above trend is that once personal information is shared with another party, it can potentially be linked to that individual forever. Thus, monitoring privacy implications of mobile apps should not focus just on a snapshot of their behavior, but also on how their behavior evolved over time. In fact, because apps are regularly updated with new versions (as frequently as once a month on average [12], [19]) that fix bugs, improve performance, add features, and even *change what is shared with other parties*, it is essential to study app behavior *across versions*.

In this paper, we are the first to conduct a comprehensive, longitudinal study of the privacy implications of using multiple versions of popular mobile apps across each app's lifetime. We focus specifically on Android apps[1] and identify when personally identifiable information (PII) appears in Internet traffic while using them. Through hybrid automated and scripted interactions with 512 apps (across 7,665 distinct versions), we compile a dataset that informs *what* information is exposed over the Internet (identifiers, locations, passwords, *etc.*), *how* it is exposed (encrypted or plaintext), and *to whom* that information is exposed (first or third party). We analyze this dataset to understand how privacy has changed over time (for individual apps and in aggregate across popular apps), why these trends occur, and what their implications are.

Our work substantially extends existing mobile privacy research [23], [43], [49], [50] by focusing on multiple versions of apps instead of individual versions. Moreover, most existing *longitudinal* studies infer privacy risks by using static analysis to monitor library usage and permission requests [12], [15], [53], [54]. In contrast, we detect actual PII transmitted in network traffic to other parties while an app is used.

Gathering a longitudinal view of the privacy implications of using apps over time poses the following challenges:

- Monitoring app behavior across versions for a large number of apps requires a methodology that scales accordingly. Manually logging into apps and interacting with them can comprehensively trigger privacy leaks, but this is infeasible at scale. Instead, we use a semi-automated approach that incorporates random interactions [33] and manually generated scripts for logging into apps.
- We need a way to identify the privacy risks for each app. To this end, we analyze network traffic[2] generated by the mobile device running the app, using both simple text matching on known identifiers and machine-learning inference [50] to extract identifiers not known in advance.
- We need a systematic, configurable, and meaningful way to compare the privacy guarantees of the apps (and their versions). To this end, we identify several metrics that provide insight into privacy trends and implications.

Using the above approach, our study is the first to reveal the privacy implications of popular apps across multiple versions:

On average, privacy has worsened over time. We analyze privacy risks along multiple attributes (what PII is leaked, to how many destinations, and whether it is encrypted) independently and in combination. We find that apps increasingly leak more types of PII and to more domains over time, but HTTPS adoption has seen slow growth. When combining these factors, we find that about a quarter of apps (26.3%) are getting better with respect to

[1]The only platform where we can access historical versions of apps.

[2]We focus only on IP traffic. A recent study [43] showed that less than 1% of leaks occur over non-IP traffic (i.e., SMS).

Network and Distributed Systems Security (NDSS) Symposium 2018
18-21 February 2018, San Diego, CA, USA
ISBN 1-1891562-49-5
http://dx.doi.org/10.14722/ndss.2018.23143
www.ndss-symposium.org

privacy, *but twice as many are getting worse over time (51.1%)*, with only a small fraction (9.5%) staying the same or exhibiting highly variable privacy risks between versions (13.1%).

Snapshots of privacy leaks from single versions of apps are incomplete. For all but 7% of the apps in our dataset, studying one version will miss PII gathered across all versions of the app. We also find that the set of PII leaked by an app changes frequently across versions.

HTTPS adoption is slow. Unlike recent trends in HTTPS adoption for web traffic [25], we find that apps are slow to adopt HTTPS. In fact, from the moment we see that a domain first starts supporting HTTPS, it takes five years for at least half of the apps in our study to start using it. Overall, the fraction of flows using HTTPS has remained nearly constant over the time period covered by our study.

Third-party tracking is pervasive. While previous work using small snapshots of time demonstrates that third parties collect substantial amounts of PII, we find the problem to be even worse when considering PII leaks across versions. We find that there is little correlation between the amount of traffic to a third party and the volume of PII it leaks. In addition, we analyze how third parties (among which several are not highlighted in previous studies) collect locations, email addresses and gender along with tracking identifiers, enabling fine-grained tracking of users and their daily activities.

In summary, our key contributions are: (1) a large-scale privacy analysis across multiple apps and app versions, (2) a dataset of network traffic generated by running apps, along with labels describing the PII contained in them, and (3) an analysis of the origins and privacy implications of these information leaks. Our data and analysis are available at `https://recon.meddle.mobi/appversions/`.

II. Related Work

A large body of related work has investigated the privacy of mobile apps and their generated network traffic. Most related studies focus on identifying personal information that is (or might be) exposed to other parties over the Internet, using one or more of the following complementary approaches.

Static analysis. This technique entails analyzing an app's bytecode using symbolic execution [58] and/or control flow graphs [10], [11], [22]. Several academic studies leverage static analysis to inspect app permissions and their associated system calls [11], [41], to audit third-party library usage [17], [51], and to analyze flaws in HTTPS usage and certificate validation [24], [27]. This approach is appealing because it enables large-scale app analysis without the overhead of running or interacting with apps. However, static analysis may identify privacy leaks in code that is rarely or never executed; further, it cannot analyze dynamically loaded code, which is commonly used to update app functionality at runtime in as much as 30% of apps [43].

Dynamic analysis. In contrast to static analysis, dynamic analysis tracks system calls and access to sensitive information *at runtime*. In this approach, the runtime (e.g., the OS) is instrumented to track memory references to private information and *taint* the memory it is copied into. This taint propagates as the information is copied and mutated; ultimately when it is copied to a sink, such as the network interface, it is flagged as a PII leak. TaintDroid [23] is commonly used for dynamic analysis of Android apps. While taint tracking can ensure coverage of all PII leaks (even those that are obfuscated), it requires some form of interaction with running apps to trigger leaks. Typically, researchers use automated "UI monkeys" [33], [44] for random exploration or more structured approaches [16], [37] to generate synthetic user actions; however, prior work showed that this can underestimate PII leaks compared to manual (human) interactions [50].

Network traffic analysis. This approach relies on the observation that PII exposure almost always occurs over Internet traffic. Thus, network traffic analysis focuses on identifying PII contained in app-generated IP traffic [40], [49], [50], [52]. The benefit of this approach is that it works across platforms without the need for custom mobile OSes or access to app source code, and thus is easy to deploy to user devices for the purpose of real-time analysis and detection of PII leaks. A drawback is that it requires the ability to reliably identify PII (which may be encrypted and/or obfuscated) in network traffic. All of the above approaches support TLS interception to access plaintext traffic to search for PII, but differ in what they search for: most on-device approaches search for known PII stored on the device [40], [49], [50], [52], whereas ReCon [50] also uses a machine-learning approach to infer a broader range of PII that includes user input. However, these approaches are susceptible to missing PII leaks from apps that defend against TLS interception, or that use non-trivial obfuscation or encryption of PII [21].

Longitudinal analysis. Some existing longitudinal studies use static analysis to study how apps across several categories [54], and finance apps in particular [53], change over time in terms of permission requests and security features and vulnerabilities, including HTTP(S) usage. Similarly, Book et al. conduct a longitudinal analysis of ad libraries [15], but they focus only on permission usage. While partially sharing the goals of our work, these studies do not actually execute and analyze apps, and thus are subject to both false positives (by looking at permissions and code paths that are not used) and false negatives (by not covering code that is dynamically loaded at runtime).

To the best of our knowledge, our study provides the first longitudinal analysis of privacy risks in network traffic generated by running app versions that span each app's lifetime. Our work complements and substantially extends the related work presented above: we study privacy across versions (and thus over time), whereas most previous work consists of one-off studies that focus on individual versions of apps available at a certain moment in time. Moreover, since we monitor the traffic exchanged by actual apps running on real devices, we overcome some of the limitations of the discussed static and dynamic analysis approaches.

III. Goals and PII Definitions

The primary goal of this work is to understand the privacy implications of using and updating popular Android apps over time. As privacy is a top-cited reason for why users do not install app updates [55], studying PII leaks from apps across versions can help users make more informed decisions. Furthermore, this information can assist regulators when auditing and enforcing privacy rules for mobile apps [26]. An explicit non-goal of this work is coverage of all versions of all apps; rather, we focus on a diverse set of 512 popular Android apps.

Unique Identifier (ID)	Advertising ID (Ad ID), IMEI, Android ID, MAC address (MAC Addr), IMSI, Google Service Framework ID (GSF ID), SIM card ID (SIM ID), Hardware serial (HW Serial)
Personal Information (User)	email address, first and last name, date of birth (DOB), phone number, contact info, gender
Location	GPS location (Location), zip code (Zip)
Credential	username, password

TABLE I: **List of PII categories and types.**

Number of APKs	7,665 (512 unique apps)
APK release timeframe	8 years
Versions per app (mean)	15.0
Versions per app (median)	14
HTTP(S) flows per app (mean)	94.7
Total HTTP(S) traffic	33.6 GB (pcap format)
Total number of flows	675,898
Unique third-party domains	1,913

TABLE II: **Dataset description.**

A. PII Considered in This Work

Personally identifiable information (PII) is a generic term for describing "information that can be used to distinguish or trace an individual's identity" [38]. In this paper, we define PII to be a subset of this, based on textual data that can be gathered and shared by mobile apps. Specifically, we consider the PII listed in Table I, which is based on a combination of PII accessible from Android APIs, user-supplied information, and inferred user information that was reported as being leaked in network traffic in previous work [40], [49], [50], [52].

B. Threat Model and PII Leaks

We define privacy risks and PII leaks in the context of the following threat model. We assume that the adversary seeks to collect PII from an app running on a user's mobile device. The adversary is any party that receives this information via network connections established when running an app, including the following:

- *App provider*, i.e., the company that releases an app, also referred to as the *first* party.
- *Other parties*, e.g., the ones that are contacted by an app as part of advertising, analytics, or other services, also referred to as a *third* party.
- *Eavesdroppers*, who observe network traffic (e.g., an ISP, an adversary listening to unencrypted WiFi traffic, or one that taps an Internet connection).

We define two goals of an adversary that motivate our definition of PII leak as a privacy risk:

Data aggregation. This occurs when first or third parties collect information about a user over time, including which apps they use, how often they use them, where they are located when they do so, *etc.* The risk from this kind of information gathering is that it can be used to build rich profiles of individuals, which can in turn be used for targeted advertising [14], price discrimination [36], and other differential treatment driven by algorithms using this information [35].

Eavesdropping. In this scenario, the adversary learns a user's information passively by observing network traffic (e.g., plaintext PII leaks). This presents a privacy risk to users in that it constitutes a third party for which the user did not explicitly consent to collect data. Furthermore, it can constitute a security risk when information exposed to unauthorized third parties includes credentials (i.e., username and password).

We define a *PII leak* as any case in which information listed in Table I is transmitted to a first or third party, with the exception of credentials that are sent to a first party via an encrypted channel. The latter is excluded because it is exclusively provided intentionally by a user. We cannot in general determine whether other cases of PII are intentionally disclosed to other parties (and/or required for app functionality), so we include them in our analysis for completeness. Note that the goal of this work is to increase privacy transparency, so we leave the decision as to what constitutes an unintentional and important leak to the users of our dataset and analysis. To this end, our interactive tool [1] allows users to set preferences for the importance of each type of leak.

IV. Methodology

We identify and analyze PII leaks using network traffic analysis on flows generated by automated and scripted interactions with multiple versions of popular Android apps. Our methodology consists of four high-level steps: (1) selecting apps for analysis, (2) collecting historical and current versions for each app, (3) interacting with these APKs (i.e., unique versions of each app), and (4) identifying and labeling PII leaks. In this section, we discuss each individual step in detail. We further discuss the assumptions, limitations, and validation of our approach. Table II summarizes our dataset.

A. App Selection

We selected 512 apps for analysis in this study, using the following criteria:

- *Popularity*. We started with the set of apps that was either in the top 600 popular free apps according to the Google Play Store ranking, or in the top 50 in each app category, as of January 10, 2017. We exclude apps that require financial accounts or verified identities (e.g., bank and credit card accounts, social security numbers).
- *Multiple versions.* We considered only apps with more than three versions compatible with our analysis testbed, which includes devices running Android 4.4.4 and Android 6.0. These OS versions run on approximately 50% of Android devices as of May 2017 [29].
- *Amenable to traffic analysis.* As discussed in Section IV-C, we collect both unencrypted (HTTP) traffic and the plaintext context of encrypted (HTTPS) traffic via TLS interception [8]. We exclude 26 apps (e.g., Choice of Love, Nokia Health Mae and Line Webtoon - Free Comics) where most versions crash or hang when opened, or that do not permit TLS interception as explained in Sec. IV-E.

B. APK Collection

After identifying apps to analyze, we gather their historical and current versions, and label their release dates.

Finding app versions. Officially, the Google Play Store only supports the download of the most recent version of each app. However, Backes et al. [12] reported an undocumented API of

the Google Play Store that allows downloads of an arbitrary version of an app (i.e., its Android Package Kit, or APK, file), as long as the app's version code[3] is known. The authors identify several patterns, which we build upon, to identify app version codes. For the 512 selected apps, we downloaded 7,665 APKs. Some apps have hundreds of versions, and testing all of them would be prohibitively expensive. Thus, for apps with more than 30 different versions, we sort their releases chronologically and pick 30 versions that are evenly distributed across all versions.

Inferring APK release date. The API that we use for downloading APKs does not provide the release date for each app, information that is essential for understanding how app behavior changes over time. To address this, we leverage the fact that developers who release a new version of an app must update the version code in several files inside the APK (*AndroidManifest.xml and* META-INF/MANIFEST.MF). We thus infer the release date based on the modification time of these files, which assumes that the developers' OS timestamps correctly. Of the 7,665 APKs we downloaded, 429 APKs had timestamps that were obviously incorrect (e.g., a date before Android's first release on August 21, 2008 or a date in the future). For these cases, we manually checked release dates with several third-party services [2]–[5] that provide release dates for the last three years.

To understand how well our heuristics work, we manually cross-validated the release dates of 77 APKs by comparing the file modification times and release dates found using the above third-party services [2]–[5]). We find that 88% of inferred release dates differ with the public record by less than a week, and only two cases have a difference of 30 days or more. We investigated these last two cases and found that the difference in release date is likely due to a developer error, not an incorrect inference. Namely, these are cases where the developer released a new version of the app without updating the version string in the APK. As a result, the date from the third-party services did not correspond to the APK we investigated. The average interval between each update across apps is 47 days, with a standard deviation of 181. Note that 21% of the 512 apps were first released before January 1st, 2012 and exactly half were released before August 22nd, 2014.

C. Interaction and Traffic Collection

In this step, we interact with each APK and collect the network traffic generated as a result from these interactions.

Test environment. We conduct experiments using five Android devices: one Nexus 6P phone and one Nexus 5X phone, both with Android 6.0.0; and three Nexus 5 phones with Android 4.4.4. We use real Android devices instead of emulated ones to avoid scenarios where apps and third-party libraries detect the analysis environment and modify their behavior accordingly. It has been shown that emulators are easy to fingerprint [47], [57], a fact that is exploited for example by ad libraries to only show ads and leak data when executed on a real device [46].

Interaction with apps. Measuring PII leaks from apps requires interacting with them, and the gold standard for doing so is via natural human interaction. However, manually interacting with each of the selected 512 apps (7,665 unique versions) is not practical. Thus, we use Android's UI/Application Exerciser Monkey [33], a tool that automatically generates pseudo-random UI interaction events (swipes, taps, *etc.*) for an app. While a number of other approaches for automation have been proposed, a recent study [18] showed that Monkey exhibited a better coverage in terms of code coverage and fault detection capabilities than other automated tools. Completely random events would prevent apples-to-apples comparison among versions of the same app, so we specify the same random seed that generates the sequence of events for interaction with all of an app's versions.[4] Specifically, we use Monkey to generate approximately 5,000 user events by specifying five seeds for 1,000 events each.[5] We use 5,000 events because it allows us to test a large number of APKs in a reasonable amount of time, and because previous work [42] found that longer interaction times do not substantially impact the set of PII that leaked. We cross-validate our dataset with human interactions in Section IV-F.

Many apps (75 in our study) require users to log in with a username and password before accessing app functionality. Thus, failure to login can severely underestimate the amount of PII leaked. We created accounts for testing with each of these apps, but manually logging into each version is prohibitively expensive. We avoided this by recording the login events in one version and replaying the events in other versions using RERAN [28]. We perform both record and replay of login actions on the same device to ensure a consistent UI layout.

Recording network traffic. For each experiment, we run one app at a time. To collect network traffic while interacting with the apps, we redirect the traffic to a proxy server that records plaintext traffic and that uses TLS interception (using mitmproxy [8]) to record the plaintext content of HTTPS requests. For apps that prevent TLS interception via certificate pinning, we use JustTrustMe [7], a tool that modifies Android in such a way that certificate validation using built-in OS libraries always succeeds. We test such apps only on devices running Android 4.4.4 (the Nexus 5 phones) because JustTrustMe does not support later OS versions.

D. Privacy Attributes

After the completion of the experiments, we analyze network traffic according to the following three *privacy attributes* to assist in our subsequent analysis of network flows.

1) PII Leaks: We label each flow with the PII that it leaks in two phases. First, we use simple string matching to identify PII that is static and known in advance (e.g., unique identifiers, personal information, zip code, and credentials). This approach, however, cannot be reliably applied to dynamic values (e.g., fine-grained GPS locations) and to data not directly input into an app (e.g., gender).

For these cases, we use ReCon [50], which uses machine learning to infer when PII is leaked without needing to rely on exact string matching. The key intuition behind ReCon is that PII is often leaked in a structured format (e.g., key/value pairs

[3] An integer value that can be incremented by arbitrary values from one version to the next.

[4] Note that we do not explicitly account for changes in UI or functionality over time because doing so requires manual analysis and is infeasible at this scale. However, we rely on the randomness of Monkey to probabilistically exercise UIs and functionality as they change.

[5] Batches of events were required to give apps sufficient time to process events; failure to do so led to crashes or exits before the events completed.

such as `password=R3Con` or `adId:93A48DF23`), and that the text surrounding PII leaks can become a reliable indicator of a leak. ReCon therefore uses a classifier to reliably identify when network traffic contains a leak (e.g., in a simple case, looking for `password=`), without needing to know the precise PII values. We manually validated all cases of inferred PII leaks to ensure their correctness.

2) Transport Security: This study focuses exclusively on HTTP and HTTPS traffic. In addition to the standard ports 80 and 443, we also include port 8080 for HTTP traffic and ports 587, 465, 993, 5222, 5228 and 8883 for HTTPS traffic. We find that only 0.5% of the flows in our dataset use other ports.

3) Communication with First and Third Parties: An important privacy concern is who receives the PII. In a network flow, this corresponds to the owner of the traffic's destination. We distinguish between *first*-party second-level domains (hereafter simply referred to as *domains*), in which case the developer of an app also owns the domain, and *third*-party domains, which include ad networks, trackers, social networks, and any other party that an app contacts. For instance, `facebook.com` is a first party to the Facebook app, but it is a third party to a game app that uses it to share results on Facebook.

Our domain categorization works in two steps. We first take all the domains that we observed in our experiments and build a graph of these domains, where each node represents a domain and each edge connects domains belonging to the same owner. We then match the owner of each connected subgraph of domains to the developer of an app and consequently label them as first-party domains for that app. Our approach is similar to related work focusing on identifying the organizations behind third-party ad and tracking services [56], which found that current domain classification lists are incomplete and too web-centric to accurately identify mobile third-party domains.

Ownership of domains. To identify a domain's owner, we leverage *WHOIS information*, which contains the name, email address and physical address of the registrant unless the registration is protected by WHOIS privacy. As a preprocessing step, we thus first discard any WHOIS entries that are protected by WHOIS privacy. We then connect domains as belonging to the same owner based on (1) the registrant's name and organization, and (2) their email address (excluding generic abuse-related email address from the registrar). This method allows us to group together disparate domains that belong to the same owner, e.g., we can identify `instagram.com`, `whatsapp.com` and `atlassbx.com` as Facebook-owned services.

Ownership of apps. To identify the developer of an app, we use information from the *Google Play Store listing*, which contains the name of the developer, and optionally their website, email address and physical address. Some developers use third-party services (e.g., Facebook pages) in lieu of hosting their own website, or free email providers, such as Gmail. We filter out such cases from our analysis. Since Google recommends using "Internet domain ownership as the basis for [...] package names (in reverse)" [30], in the simplest case the package name embeds one of the developer's domains. Otherwise, we compare the developer information from Google Play against WHOIS records for a domain as detailed below.

First-party identification. We identify traffic to a domain as *first party* when information about the owner of the domain matches information about the owner of an app. We label any domain collected from the app's Google Play Store listing as first party, as well as the domain in the app's package name. We also label as first party any domains that are registered to the same name, organization, physical address, or email address as the ones listed for the developer in Google Play. To account for any inconsistencies in the representation of the physical addresses, we first convert them with geopy [6] to their coordinates through the Google Geocoding API [31].

Third-party identification. We label as third party all the domains that have not been labeled as first party according to the previous paragraph. This includes ad and tracker domains, content hosting services or any third-party domain an app contacts to fetch content.[6] Our classification is skewed towards finding potential third-party services; we validate parts of our approach in Section IV-F.

E. Assumptions and Limitations

Our approach uses several assumptions and heuristics to inform our longitudinal analysis of privacy across app versions. We now discuss these assumptions and the corresponding limitations of our study.

Coverage. We do not cover all apps or all app versions, but rather focus on a set containing many versions of popular apps across multiple categories of the Google Play Store. We believe this is sufficient to understand privacy trends for important apps, but our results provide at best a conservative underestimate of the PII exposed across versions and over time.

TLS interception. TLS interception works when apps trust our self-signed root certificates, or when they use built-in Android libraries to validate pinned certificates. We are also constrained by JustTrustMe. As a result, we cannot intercept TLS traffic for 11 apps that possibly use non-native Android TLS libraries (e.g., Dropbox, MyFitnessPal, Snapchat, Twitter) [48].

Obfuscation. Due to the inherent limitation of network traffic analysis, we do not detect PII leaks using non-trivial obfuscation, as it requires static or dynamic code analysis. In such cases, we will underestimate PII leaks. However, we do handle non-ASCII content encodings and obfuscation. For the former, we examine the *Content-Encoding* field in the HTTP header, and decode gzip flows (2.5% of total flows). We further decode content using Base64 but did not find any additional leaks using this encoding. For the latter, we apply standard hash functions (MD5, SHA1, SHA256, SHA512) on our set of known PII, and match on the result. This yielded 4,969 leaks (4.3% of all leaks observed in this study) in 4,251 flows.

Testing old versions today. We assume old versions of apps exhibit the same behavior today as when they were initially released. However, for a variety of reasons (e.g., different behavior of the contacted domain, or domain being no longer available), this might not always be true. It is likely that this means we will underestimate the PII leaked by apps (e.g., if a domain does not resolve to an IP).

[6]This includes domains provided to their customers by Google App Engine or Amazon Web Services. We argue that even if the services running on these platforms belong to a first party, communication to these platforms should still be considered third-party communication because developers do not have ownership of, or full control over, the platform.

Because we could not run old versions of these apps at the time they were released, we must use heuristics to determine whether our analysis might be impacted by such factors. During the course of our experiments, we found that the behavior of leaks and domains contacted did not change significantly over several months; as such, we do not think this is an issue for recently released app versions.

For older versions of apps, we assume that DNS and HTTP failures potentially indicate apps that no longer function similarly to when they were first released. Thus, we exclude APKs for which more than 10% of DNS requests fail or 10% HTTP responses are error codes (4xx or 5xx response codes). This was the case for 15 apps (2.8% of the original selected apps).

F. Validation

To improve confidence in the accuracy and representativeness of our measurements, we validated several critical aspects of our approach as follows.

Automated interaction. A limitation of automated interactions with apps is that they may not elicit the same privacy-related behavior as user interactions. To estimate this gap, we compare our results with those from the natural human interactions made available by the Lumen [49] project, which provides on-device privacy analysis and has thousands of users. Lumen maps network flows to the source APKs and destination domains, and also labels any PII that matches those stored on the device. We found 983 APKs that appear in both our and Lumen's datasets and 380 of which leaked PII in both studies. The latter corresponds to 122 distinct apps (24% of the 512 apps in this study) that cover 23 app categories. On average, our dataset missed 0.41 PII types per APK found by Lumen, with a range of 0–3 missing types from automated tests. The most frequently missed types include Android ID (52%), email (15%), MAC address (12%) and IMEI (11%). Similarly, the number of unique domains and protocol pairs per app missed by our automated tests compared to Lumen is 2.36 (standard deviation of 4.42). On the other hand, Lumen missed on average 1.38 PII types per APK that our approach found (with a range of 0—6 types). The most common missed types are advertiser ID (27%), hardware serial (18%), Android ID (15%) and Location (15%). *In summary, human interactions find different PII leak types and traffic to different domains, as expected; however, the gap between these two datasets is relatively small on average.* As a result, we believe our analysis covers most of the behavior one would expect to see in the wild.

Repeatability. A potential problem of our study is that our automated tests use only one round of 5,000 interaction events for each APK. It is unclear *a priori* whether this approach will yield similar results over time, and thus might be biased in some way. To test whether this is the case, we repeated the experiments for the five apps (105 APKs) that have a large variance in leaked PII types across versions. In particular, we performed a pairwise comparison between the PII types leaked by different versions of each app and selected the apps with the largest number of distinct sets of PII types across versions. For each APK, we performed the same experiment each day at approximately the same time of day, for ten days. After we collect the traffic ten times, we compare the number of unique PII leak types, the number of domains contacted, and the fraction of flows using HTTPS. *We find that the change in results over repeated experiments is small*: for more than 90% of tested APKs, the variation across experiments is generally no more than one PII type, two domains, and a fraction of HTTPS traffic of no more than 6.0%.

Domain categorization. Our approach to distinguish between first-party and third-party domains largely relies on WHOIS data, which is known for its incompleteness and noise. To validate our approach we manually verified the domain classification for a subset of 20 apps, which we selected randomly from all apps that leak PII and contacted more than one domain in our experiments. We inspected 550 app/domain pairs (343 unique domains), 60 of which our approach labeled as first-party domains and the remaining 490 as third-party domains. *We find that all of these first-party labels are indeed correct, with only a small number of false negatives:* our approach missed 5 first-party domains for 3 apps. Overall, we find those results encouraging as our study is focused on analyzing third-party services.

V. Longitudinal Analysis

This section presents our analyses and findings regarding changes in PII leaks across app versions and time. Section V-A presents the case of a single notable app (Pinterest). In Section V-B, we provide a summary of all the PII leaked across all APKs in our dataset. Section V-C focuses on how specific types of PII are leaked over time for each app. We analyze trends in HTTPS adoption and third-party destinations in Sections V-D and V-E. Section V-F summarizes our key findings.

A. A Notable Example: Pinterest

To demonstrate our analysis of privacy attributes, we use the Pinterest app as an in-depth example. In Figures 1a and 1b we show how PII leaks and network flows with third-party services change in the Pinterest app across different versions.[7] In the plots, each app version is identified by a different *version code* on the x-axis, sorted in ascending chronological order.

Figure 1a shows how many times each PII type is leaked across all network flows for each version, where the y-axis for each time series represents the number of times it is leaked during an experiment. The number below the PII type is the maximum number of times any version of Pinterest leaked the PII type. The stacked bars are colored according to the domain type and protocol. The plot shows that the app sends user passwords to a third party[8] and starts leaking gender, location, advertiser ID and GSF ID in more recent versions. In addition, the frequency of Android ID leaks increases by two orders of magnitude.

B. Summary of Results

This section focuses on a summary of PII leaked across all versions (APKs) of all apps that we tested, and their implications for privacy risks over time.

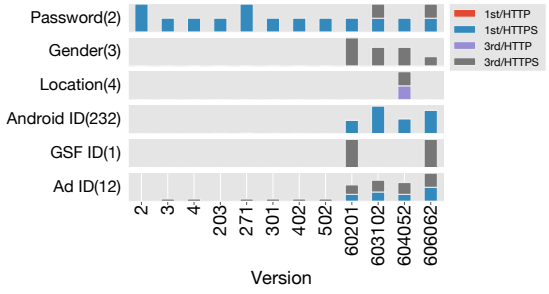
Table III depicts our results, where each row is a PII type, and each column counts the number of instances falling into a given category. The table is sorted in descending order according to the number of apps leaking each type.

[7] Similar plots for every app in our dataset can be found online [1].

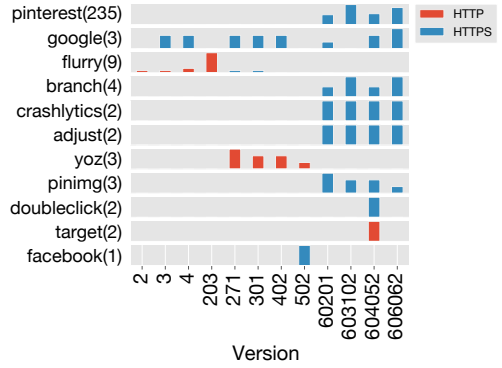
[8] We responsibly disclosed this security bug, which Pinterest confirmed and fixed in later versions not included in this study.

PII Type	Leaks Overall				Leaks to First Party				Leaks to Third Party			
					HTTP		*HTTPS*		*HTTP*		*HTTPS*	
	#Apps	%Apps	#APKs	%APKs	#Apps	#APKs	#Apps	#APKs	#Apps	#APKs	#Apps	#APKs
Ad ID	314	62.2%	2,270	29.8%	23	115	32	227	149	700	282	2,037
Location	268	53.1%	1,577	20.7%	27	258	41	301	96	450	209	778
HW Serial	254	50.3%	1,157	15.2%	10	81	21	154	28	170	227	832
IMEI	167	33.1%	1,597	21.0%	45	443	32	250	62	505	123	1,073
Android ID	124	24.6%	1,225	16.1%	18	163	28	272	54	423	104	957
GSF ID	108	21.4%	504	6.6%	0	0	9	68	0	0	99	436
MAC Addr.	71	14.1%	649	8.5%	8	105	12	116	38	307	25	173
Gender	65	12.9%	257	3.4%	6	68	5	16	35	106	42	134
Email	43	8.5%	280	3.7%	12	97	21	124	3	19	14	58
Password	13	2.6%	84	1.1%	6	48	N/A	N/A	0	0	7	36
Last Name	6	1.2%	37	0.5%	0	0	2	15	0	0	4	22
First Name	6	1.2%	37	0.5%	0	0	2	15	0	0	4	22
PhoneNo.	3	0.6%	18	0.2%	0	0	2	15	0	0	2	7
SIM ID	2	0.4%	9	0.1%	2	9	0	0	0	0	0	0
Any PII Type	*505*		*7611*		-	-	-	-	-	-	-	-

TABLE III: **Summary of PII types leaked by apps/APKs, sorted by number of apps.** The majority of apps and APKs leak at least one PII type. The fractions for the APKs are significantly lower than the ones for the apps, indicating that not every version leaks PII. Unique IDs and locations are the most common leaks across apps. Unique IDs are leaked to third parties much more often than to the first party, given the free monetizing model using ads. We also found *13 cases of password leaks*.



(a) PII types by destination type and protocol.



(b) Domains by protocol.

Fig. 1: **Example app privacy attributes for Pinterest**. The x-axis corresponds to chronological versions of the app. In (a), the y-axis of each stacked bar plot is the number of times a version leaks a PII type, and the bar plots are colored according to the domain type and the communication channel; in (b), the y-axis of each stacked bar plot is the number of times a version contacts a domain, and the bar plots are colored according to protocol. For (a) and (b), the number in parentheses to the left of each y-axis is the maximum y value across all versions. Similar plots for other apps can be found on our website [1].

The first two columns show the number of apps and APKs leaking each PII type. In line with previous work [50], we find that the most commonly leaked PII types are unique identifiers (more than half of all apps leak an advertiser ID and/or hardware serial number) and locations (53.1% of apps). We nonetheless still find a substantial fraction of apps (more than 10%) leaking highly personal and security-sensitive information such as email addresses (often to analytics services such as `kochava.com` and `crashlytics.com`), phone numbers (e.g., collected by `crashlytics.com`, `segment.io`, and `apptentive.com`), and gender. However, when focusing on APKs (2nd column), we find that substantially lower fractions leak each PII type—indicating that most PII types are not leaked in every app version. We explore this phenomenon in more detail in Section V-C. In the table we can also see that there are 13 apps leaking *passwords*: 6 apps leak passwords in plaintext, and 7 apps send passwords to third-party domains. Of these apps, in the latest version we tested (not shown in the table), we discovered that 4 apps still leak plaintext passwords (*Meet24, FastMeet, Waplog, Period & Ovulation Tracker*).[9]

The next group of columns focuses on the number of apps and APKs leaking each data type to a first party, either via HTTP or HTTPS. Here we find that there is no clear pattern for HTTPS prevalence for PII leaks to first parties, except for a clear (and easily explained) bias toward password encryption. When compared with the third column group ("Leaks to Third Party"), it is clear that the vast majority of instances of PII leaks go to third parties (with the exception of passwords, with small but nonzero occurrences). This is likely explained by the fact that PII is typically harvested to monetize users via targeted ads, often over HTTPS. This result is a double-edged sword: encryption improves privacy from network eavesdroppers, but it also frustrates attempts by stakeholders (e.g., users, researchers, and regulators) to audit leaks.

To understand whether certain categories of apps are relatively better or worse for privacy, we grouped them by category as reported in the Google Play Store.[10] Table IV provides results for the top five and bottom five categories in terms of the average number of PII types that are leaked by apps in the category. We find that the categories that leak the largest number of

[9] We responsibly disclosed these leaks to the developers (multiple times over a period of months) and received no response.

[10] We only used the category of the most recent version of the app we tested, even if the app was assigned a different category in a previous version.

App Category	Apps	APKs	#PT	#PI	#3PD	%S
Food & Drink	2	50	2.9	26.3	7.1	52.7
Dating	6	108	2.3	38.4	10.0	60.7
Lifestyle & Beauty	9	137	2.0	40.9	10.7	65.7
Games	76	1231	2.0	70.8	9.7	61.2
Finance	3	28	1.9	42.3	8.2	96.8
...						
Auto & Vehicles	7	122	0.8	4.6	8.8	84.9
Weather	10	177	0.8	88.5	7.3	47.7
Libraries & Demo	4	51	0.7	29.6	4.1	82.2
Art & Design	6	101	0.7	7.7	5.2	69.3
Events	6	104	0.6	7.9	5.7	95.6

TABLE IV: **Average privacy attributes per app category, sorted by number of unique PII types (PT) leaked.** Only the top and bottom five categories are shown. **PI** refers to the number of instances of PII leaks, **3PD** refers to the number of second-level third-party domains contacted, and **S** refers to the fraction of HTTPS flows. *Dating* and *Food & Drink* apps are among the worst in terms of number and types of PII leaks, and these substantial fractions of their flows are unencrypted.

PII types or cases (and contact the most third-party domains) include Lifestyle & Beauty, Games, Finance, Entertainment and Dating, while Art & Design and Events leak the fewest. With the exception of Finance, the apps that leak the most PII types also send a significant fraction of their traffic (34–47%) without encryption, thus exposing PII to network eavesdroppers.

C. Variations in PII Leaks

Since privacy risks across versions of an app rarely stay the same, a study that looks into a single version of an app is likely to miss PII leaks affecting a user that regularly uses and updates the app. In this section, we first quantify how many PII leaks previous work may miss by focusing on one version, and then we quantify how the frequency of PII leaks changes across versions and time.

PII leaks across versions. In Figure 2a we show the CDF describing the minimum, average, and maximum number of distinct PII types leaked by individual apps across all their versions (*Min*, *Average*, *Max* curves); and the CDF describing the number of distinct PII types leaked during the whole lifetime of the app (i.e., the union of its versions – *Union* curve). By looking at the plot, we find a substantial gap between the maximum number of PII types leaked by an app version and the minimum, validating our hypothesis that a study using a single version of an app is likely to miss a substantial number of PII leaks. Even when focusing only on the version of an app that leaks the most PII types (*Max* curve), there is a substantial fraction of cases (37%, not shown in the figure) that miss at least one type of PII leaked by a *different* version. The average curve is strictly to the left of the union curve, indicating that a study using an arbitrary app version is likely to miss at least one type of PII. *In summary, for all but 7% of the apps in our dataset, a study using only one version is guaranteed to underestimate the PII gathered over the lifetime of the app.*

Privacy severity and changes over time. The previous analysis shows that PII leaks change over time, but do not give a clear picture of whether these changes lead to greater or less privacy risk for users. We propose addressing this by assessing the risk of PII leaked according the *severity* of each leaked type. We begin by assigning PII types to n groups, each of which has similar severity. These groups can be represented as an n-dimensional bit vector; for each APK we set the mth most significant bit to 1 if the APK leaks PII with severity m; we set the bit to zero otherwise. Importantly, when this vector is interpreted as an integer, it follows that privacy is getting worse if the integer value increases between versions, better if it decreases, and is unchanged if the value is the same.

To provide an example of how this representation informs our analysis, we use the categories of PII in Table I and define PII severity levels in the following order (from highest to lowest): password (plaintext or to a third party), username, personal information, geolocation, unique identifier. For example, consider Pinterest (Fig. 1a). Version 2 has a vector of `00001`, version 60201 has a vector of `00101`, 603102 has `10101`, and 604052 has `00111`. Note that we picked these values because they seemed reasonable to us; however, our online interactive tool [1] allows individuals to explore different relative severity levels and their impact on whether privacy is getting better or worse.

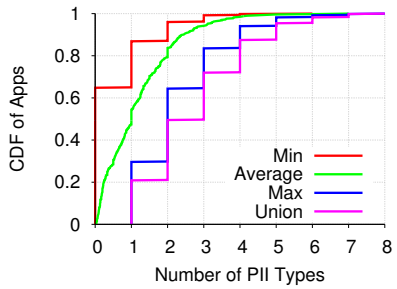
Figure 2b shows a CDF of every APK's PII severity score based on this bitmap representation. We find that nearly two thirds of APKs leak PII, but almost half of those leak only unique IDs. We also find a small fraction of APKs leaking very sensitive information such as passwords ($x > 15$). To understand how the severity of PII leaked by each app changes over time, we find the slope of the linear regression of these scores for the time-ordered set of APKs belonging to the same app. If the slope is positive, PII leak severity increased over time, negative means it decreased, and values of zero indicate no change. Figure 2c shows a CDF of these slopes for each app. The results indicate leak severity is more likely to increase (43.6%) than decrease (36.4%), and does not change for a fifth of apps.

Frequency of PII leaks. The previous paragraphs covered how many versions leaked each PII type at least once, but not how *frequently* each version leaked it. This is an important distinction because frequently leaked PII can heighten privacy risks—whether it is fine-grained location tracking over time, or increasing opportunities for network eavesdroppers to learn a user's PII from unencrypted traffic. Our analysis is in part motivated by findings from *Harvest*, a documentary film that used ReCon [50] to identify PII leaked over the course of a week from a woman's phone.[11] Specifically, her GPS location was leaked on average once every two minutes by the Michaels and Jo-Ann Fabrics apps. This behavior, thankfully, was isolated to one version of the apps; however, it raises the question of how often such "mistakes" occur in app versions.

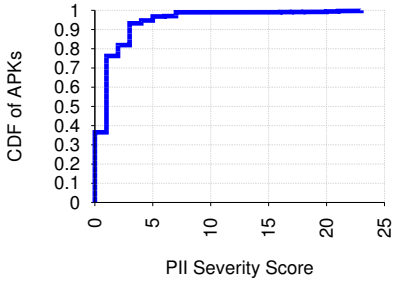
To explore this issue, we first investigate the average frequency (i.e., number of times) that each PII type is leaked by an app over time (Table V). For each app that leaks a given PII type, we calculate the mean and standard deviation of the number of times each PII type leaks across versions. The table shows that Android ID, Location, and Advertising ID are leaked most frequently on average, and also see the largest variance in terms of the number of times they are leaked.

We further investigate whether there are cases of egregious volumes of PII collection. To isolate this behavior, we calculate the difference between the minimum and maximum number of

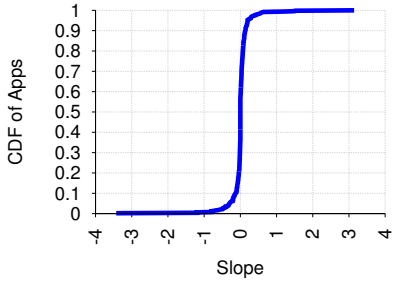
[11] `https://www.harvest-documentary.com`



(a) **Number of unique PII types per app.** Minimum, average, and maximum number of PII leaks across versions; and size of the union of PII leaked across versions, as defined in Section V-C.

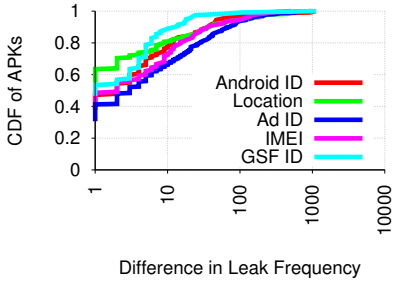


(b) **PII severity score per APK**. Most APKs leak at least one PII type; fortunately, high-severity PII leaks are rare.

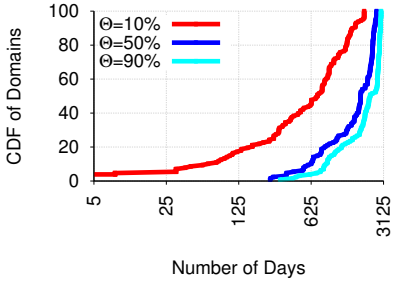


(c) **PII severity score trend per app**. Positive values (43.6% of apps) indicate that leak severity increases over time, negative values (36.4%) indicate the severity decreases. We see no change for 20% of apps.

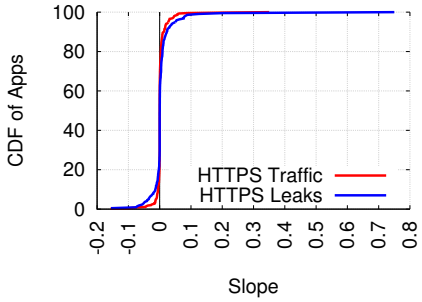
Fig. 2: **Privacy trends by PII type and severity across versions and over time.**



(a) **Largest frequency difference (logscale) per app.** There is a substantial fraction (5.6%) of apps that exhibit a several orders of magnitude difference in the frequency of PII leaks across versions.



(b) **CDF of days over Θ% of apps to adopt HTTPS per domain.** Apps adopt HTTPS extremely slowly: for half of the domains, it takes over two years for only 10% of apps to adopt HTTPS; and five years for over 50% of apps.



(c) **CDF of the slope of %HTTPS of traffic/leaks per app.** Positive values indicate an increase from the first version; negative values indicate the opposite. HTTPS adoption does not change much for most apps over time, and there is no clear trend showing increased adoption.

Fig. 3: **Privacy trends by PII frequency and HTTPS adoption across versions and over time.**

PII Type	#Apps	Mean (Mean)	Standard Deviation (Mean)
Ad ID	286	16.29	12.42
Location	256	20.33	11.63
Android ID	119	12.70	9.53
MAC Addr.	56	6.27	6.16
IMEI	140	7.87	5.68
GSF ID	109	8.55	5.37
Email	36	7.42	2.99
Gender	63	4.99	1.81
Password	13	2.48	0.92
HW Serial	240	2.16	0.76

TABLE V: **Frequency of apps leaking each PII type, sorted by the mean of the standard deviation.** For each app, we calculate the mean and the standard deviation of the number of times each PII type leaks across versions. We show the mean of both across apps. The table shows that location and unique IDs are the most tracked information, and that the number of times they leak takes on a wide range of values.

times each PII is leaked for each version, across all versions of an app. Figure 3a shows the CDF of this difference over all apps in our dataset. While the majority of apps see small differences in the frequency of leaks, there is a substantial fraction (5.6%) that exhibit a several orders of magnitude difference. To put this in context, some versions of apps leak PII once every 1 to 10 seconds on average during an experiment. Example apps include AccuWeather, Learn 50 Languages, Akinator the Genie FREE, and JW Library, which leak either location or unique ID, or both, nearly constantly.

In summary, not only are the types of PII leaks changing across versions, but also the number of times it is leaked over short periods of time. This has significant privacy implications for users who do not want their online activity and locations tracked with fine granularity.

D. HTTPS Adoption Trends

Given developments in the US and abroad concerning privacy, including reports of widespread Internet surveillance [13] and recent legislation permitting ISPs to sell user information gleaned from network traffic [45], there has been a push to encrypt Internet traffic to the greatest extent possible. Given the vast amount of personal information stored on mobile devices, HTTPS adoption by mobile apps can be perceived, at first, as a positive industry move. In this section, we investigate the extent to which apps adopt HTTPS across versions.

Aggregate results. We begin by studying the extent to which apps (across all versions) exclusively use HTTP and HTTPS, or some combination of the two. We group results according to

Party	App/Domain Pairs (#Apps)	HTTP	HTTPS	Both
All	12,143 (505)	3,559 (29.3%)	6,791 (55.9%)	1793 (14.8%)
First	703 (338)	268 (38.1%)	225 (32.0%)	210 (29.9%)
Third	11,440 (502)	3,291 (28.8%)	6,566 (57.4%)	1583 (13.8%)

TABLE VI: **Summary of domains by protocol.** The domains are separated into those that use HTTP only, HTTPS only, and both protocols. The majority of all flows use HTTPS, but this is largely due to communication with third-party sites. Substantial fractions of domains see flows without encryption and only a third of first party domains exclusively use HTTPS.

the destination second-level domain. Table VI shows the results of our analysis for all domains, as well as those previously identified as either first or third party. Across all app/domain pairs, we see that HTTPS-only adoption is the dominant behavior, with substantial fractions of app/domain pairs that use HTTP, and a relatively small fraction that use both HTTP and HTTPS for the same domain. The latter case is particularly interesting, because we know the domain supports HTTPS but for some reason some of the connections are established using plaintext.[12]

When focusing on first- versus third-party communication, we find that most of the HTTPS adoption comes from traffic to *third-party domains*. In contrast, first-party domains are nearly evenly distributed across the three categories. It is not clear why third parties use encryption more often, but reasons might include improving privacy from eavesdroppers, ensuring integrity against man-in-the-middle attacks, or making it more difficult to audit the information they gather. Likewise, the increased prevalence of mixed HTTP(S) usage for first-party domains might be due to reasons such as scarce resources for handling TLS connections, lack of need to secure content transfers, and/or mismanagement from small operators.

Speed of HTTPS adoption. We now focus on the domains that we know support HTTPS because we saw at least one flow from one APK that uses HTTPS for that domain. Once a domain supports HTTPS at a given date, we expect that any APKs contacting that domain in the future should be able to use HTTPS. However, there are many reasons why HTTPS adoption may not occur immediately for all other apps (e.g., due to using old versions of third-party libraries, or due to policy decisions to limit use of HTTPS). In Figure 3b, we investigate how long it takes a certain fraction ($\Theta\%$) of apps to adopt HTTPS for a domain, relative to the first day the domain supports HTTPS. The graph clearly shows that HTTPS adoption in mobile apps is *exceedingly slow*: for half of the domains we studied, *it takes more than two years for only 10% of apps to adopt HTTPS*. To achieve 50% HTTPS adoption ($\Theta = 50\%$ curve), it takes *five years* from the moment the domain starts supporting HTTPS.[13] This is in stark contrast to web traffic, where the only requirement for widespread HTTPS adoption is that the server supports TLS and makes it the default way to access the site.

The key take-away is that improving privacy for the content of app-generated traffic through HTTPS adoption is a slow process. This may explain why recent efforts by app stores to require HTTPS by default (or otherwise discourage HTTP use) [20], [39] have faced delayed enforcement [9].

Fraction of HTTPS traffic over time. While the previous paragraphs focus on how long it takes apps to start using HTTPS, we now focus on the question of the *fraction of app-generated traffic* using HTTPS over time. We analyze this by producing a time series of the fraction of flows that use HTTPS across versions of each app in our study. We then find the slope of the linear regression of this fraction for each app, and plot the CDF of these values as the *red* line in Figure 3c. Positive values indicate an increased fraction of HTTPS traffic over time for an app, while negative values indicate a decreased fraction. The figure shows two key trends. First, most of the values are near zero, indicating that HTTPS adoption does not change much over time. This is consistent with our results above. Second, with the exception of outliers, the number of apps that use more and less HTTPS over time are essentially equal—implying no evidence to support an increasing overall trend of HTTPS adoption as seen in web traffic [25].

A particular concern for plaintext traffic is when it contains users' PII, as they might be exposed to eavesdroppers in addition to the destination domain. We now investigate whether, over time, apps are increasingly using HTTPS when flows contain PII, to mitigate this additional privacy risk. Similar to the previous analysis, we do this using the slope of the linear regression for the fraction of PII leaks over HTTPS across versions of an app. The *blue* line in Figure 3c plots the CDF of this slope over all apps. Again, we find that the dominant trend is that HTTPS adoption does not change much over time, even for PII leaks.

E. Third-Party Characterization

In this section, we focus on the third parties that gather PII from apps, what information they gather across all apps in our study, and the implications of this data collection.

Summary of PII leaks. We now focus on the information gathered by third parties across all apps and versions in our study. We summarize our findings in Table VII, which shows information about PII leaks to third-party domains, sorted by the number of unique PII types gathered across all APKs. We show only the top 10 domains due to space limitations.

The table highlights a variety of domains that engage in broad-spectrum user tracking, usually focusing on unique identifiers, but also including sensitive information such as phone numbers and locations. Interestingly, there is little correlation between the number of flows to a domain and the number of those flows containing PII. For example, `vungle.com` leaked PII in 780 out of 1,405 flows, while `doubleclick.net` (one of the most frequently contacted domains) leaked PII in only 5% of its flows (not shown in the table). The table also shows that many domains receive more than one type of tracking identifier (e.g., Ad ID, Android ID, IMEI, GSF ID, IMEI), which allows them to continue to uniquely identify users even if the Ad ID is reset by a user. Other third-party domains, such as CDNs, are frequently contacted, but do not receive PII (e.g., `fbcdn.net`, `idomob.com`, `ytimg.com`).

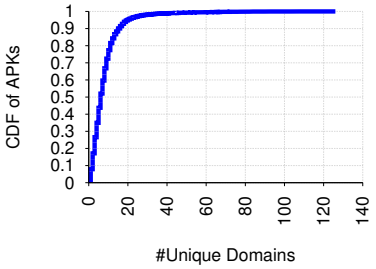
Domains contacted over time. In addition to studying the PII leaked to each domain, it is important to understand how many domains apps contact over multiple versions and how this

[12] e.g., the overhead of maintaining and establishing TLS connections, to permit caching of static content, or because HTTP URIs are hard-coded in apps.

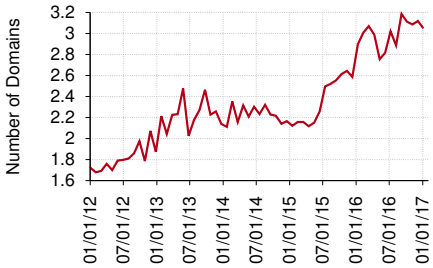
[13] The curves for Θ=75% and 90% are nearly identical to 50%.

Domain	#Flows	#PII Leaks	#Apps	# APKs	PII Types
google[*]	170,374	22,383	369	1937	HW Serial, Location, IMEI, Ad ID, GSF ID, Android ID, Gender, MAC Addr., First Name, Last Name
crashlytics.com	6,653	1,146	110	621	Ad ID, Android ID, PhoneNo., HW Serial, Email, IMEI
vungle.com	1,405	780	21	132	Ad ID, Location, Android ID, HW Serial, MAC Addr., Gender
adjust.com	1,186	650	31	176	Ad ID, Android ID, IMEI, Password, HW Serial, MAC Addr.
supersonicads.com	791	613	9	36	Ad ID, HW Serial, IMEI, Location, Android ID, MAC Addr.
amazon-adsystem.com	1,315	438	15	71	MAC Addr., HW Serial, Android ID, IMEI, Ad ID, Location
kochava.com	633	338	21	80	Android ID, Ad ID, IMEI, Email, MAC Addr., Gender
tapjoyads.com	5,503	5,390	43	440	IMEI, MAC Addr., HW Serial, Android ID, Ad ID
mopub.com	7,560	3,657	38	235	Ad ID, Android ID, Gender, Location, IMEI
applovin.com	5,591	2,360	26	149	Ad ID, Android ID, IMEI, Gender, Location

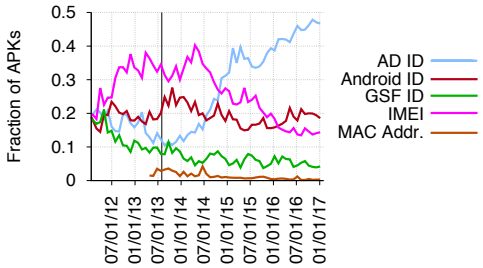
TABLE VII: **Top 10 third-party domains by flows and leaks across all apps, sorted by the number of PII types, then the number of PII leaks** (see full table online [1]). Third-party domains track mostly unique identifiers and there is little correlation between the total number of flows and the number of flows containing PII. We group the following domains as google[*]: google.com, googleapis.com, doubleclick.net, google-analytics.com, gstatic.com, googleusercontent.com, googleadservices.com.



(a) **Number of unique domains per APK**. The vast majority of APKs contact more than one domain, and more than a quarter contact 10 or more domains.



(b) **Time series of average number of unique domains receiving PII leaks, bucketed by month**. The number has nearly doubled since 2012, indicating that users' mobile activities are increasingly monitored by several parties.



(c) **Time series for fractions of APKs leaking one type of unique ID, bucketed by month.** At around 2014 (vertical line), when Google enforced the use of the Ad ID, the use of other IDs (Android ID, GSF ID, IMEI, MAC Address) decreased.

Fig. 4: **Privacy trends by domain and tracking identifier across versions and over time.**

Domain	PII Types leaked with ID	#Apps	#APKs
google[*]	Location, Gender, First Name, Last Name, Email	124	387
kochava.com	Email, Gender	8	36
vungle.com	Location, Gender	7	34
mopub.com	Gender, Location	6	13
doubleverify.com	Location	5	7
aerserv.com	Location	4	10
smartadserver.com	Location	3	7
aniview.com	Location	3	7
mmnetwork.mobi	Location	3	9
56txs4.com	Gender	3	11

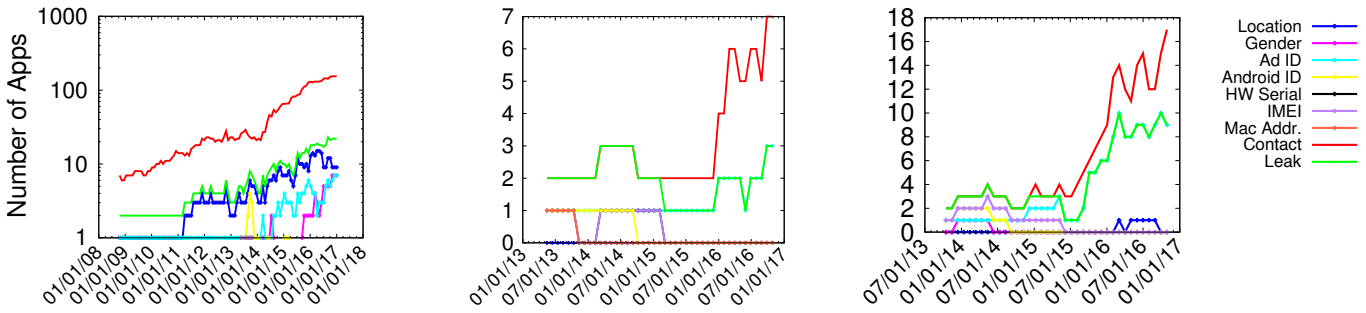
TABLE VIII: **Top 10 domains conducting high-risk tracking** (see full table online [1]). There are several domains that track non-ID PII along with unique IDs. The google[*] entry represents the same domains as specified in Table VII.

changes over time. Figure 4a shows a CDF of the number of domains contacted by each APK; we find that the vast majority of APKs contact more than one domain, and approximately one quarter of them contact 10 or more domains. To understand how this behavior changes over time Fig. 4b presents a time series of the average number of domains contacted by APKs, grouped by release date. Most notably, we find that this average has nearly doubled since 2012, with substantial increases in just the past two years. Thus, not only are large amounts of PII exposed to other parties, but each user's activity in an app tends to be tracked by more parties.

High-risk tracking. Some third-party domains track both unique identifiers and other more personal information like location, email address and gender, which allow the domain to link individuals and personal information (including locations of interest such as home, work, *etc.*) to tracking identifiers. In other words, even if a third party makes a link between unique ID and a sensitive piece of personal information *once*, it can tie this personal information to unique ID without collecting the former in the future. This is particularly problematic for user privacy, since it erodes their ability to control how they are monitored and allows cross-app tracking.

We extracted the set of domains that tie tracking identifiers with other personal information and list the top 10 (out of 95) in Table VIII. Not surprisingly, common advertising domains such as Google-owned domains doubleclick.net, googleapis.com, googleadservices.com appear at the top of the list. In addition, we find high-risk tracking from other domains, such as startappservice.com, doubleverify.com, and smartadserver.com.

Tracking identifier variations over time. In line with Google's requirements for new apps to use the user-resettable Ad ID for tracking users instead of persistent identifiers, such as the IMEI and Android ID, with enforcement of the Ad ID for new and updated apps in the Play Store starting in August 2014 [32], [34], we found it led to more apps using Ad ID instead of other identifiers (Figure 4c).



(a) doubleclick.net (logscale) (b) supersonicads.com (c) applovin.com

Fig. 5: **Third-party domain PII leaks.** Each graph represents a time series for a selected domain, with data aggregated into one-month buckets. We depict the number of apps that contact the domain in *red*, and the number of apps leaking to the domain in *green*. The other lines represent the number of apps leaking the corresponding PII type to the domain. Over time, more apps leak PII to each of these domains; further, in the case of doubleclick.net the number of PII types being leaked has increased.

Per-domain tracking variations over time. We now investigate the time-evolution of how domains track various PII types, using a case study of three examples: the frequently contacted domain doubleclick.net, the less-frequently contacted applovin.com, and the rarely contacted supersonicads.com.[14] For each of these domains, we determine the number of apps that send PII to them during each month, and plot this in Figure 5. In line with our previous results, we see variations not only in the number of apps that send a given type of PII to a domain, but also which PII types are sent. Figure 5a shows that doubleclick.net started transmitting gender in 2014. In the same year, it briefly collected IMEI, HW Serial, and Android ID, then stopped doing so. We see similar behavior for supersonicads.com (Figure 5b) for three of its gathered PII types (IMEI, HW Serial, and Android ID); additionally, they stopped collecting MAC address in 2014. Finally, applovin.com collected users' gender until 2014.

In summary, we find that an important factor for higher privacy risks over time is the increased number of third-party domains that are contacted by apps and that receive PII.

F. Summary and Discussion

We analyzed app privacy leaks over time across three dimensions (PII leaks, HTTPS adoption, and domains contacted) independently, and found that by most measures app privacy is more often getting worse as users upgrade apps. In the next section, we explore combinations of these dimensions and their implications for privacy.

We showed that *a single version of an app is not enough to assess its privacy over time*. This motivates the need for continuous privacy monitoring across versions of apps as they appear. To this end, we will make our data and analysis code publicly available, and investigate how to fully automate our experimental testbed.

Our analysis shows that *HTTPS adoption is slow* for mobile apps. This exposes users' app interactions, and potentially PII, to a larger set of network observers. The problem is often challenging to fix because it might require changes both at servers (to support HTTPS), and in the app code and/or the libraries they include (to use HTTPS).

[14]We focus on three due to space limitations; more examples are online [1].

Finally, we found that as users interact with apps over time *a large number of domains are able to gather and link significant amounts of users' PII*. This highlights the need to understand how other parties gather PII longitudinally, and motivates the need for tools that allow users to limit this data collection.

VI. Multidimensional Analysis

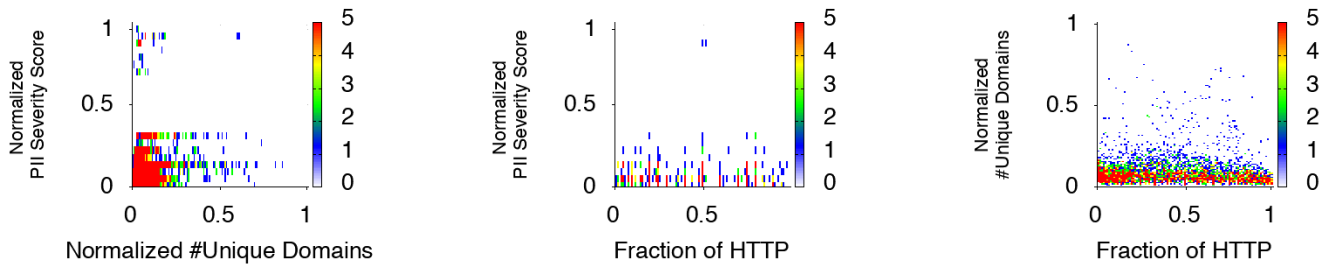
The previous sections analyzed privacy one attribute at a time; here, we focus on an APK's privacy implications when considering a *combination* of privacy attributes. For example, such analysis can indicate that an app leaking PII over insecure connections is riskier than one leaking the same PII over encrypted connections.

In the next section, we formalize the three privacy risk dimensions we consider in our multidimensional analysis. We then analyze their combination in Section VI-B. Finally, in Section VI-C we present a tool that can help individuals to visualize our dataset and understand app privacy risks in a user-friendly way.

A. Privacy Risk Dimensions

The privacy risk dimensions we consider in our multidimensional analysis are based on the privacy attributes introduced in Section IV-D, but normalized as real number between 0 and 1, with 1 indicating the highest privacy risk. Table IX shows the formal definition of each of them. For each APK j from app i ($a_{i,j}$) in our dataset, we define: (*i*) *PII type risk* $R_{i,j}$, based on the bit vector representation in Section V-C; (*ii*) *Destination domain risk* $D_{i,j}$, as the sum of the flows that leak to third-party domains divided by the maximum number of flows generated by an APK of app i; (*iii*) *Protocol risk* $P_{i,j}$, as the percentage of flows that are sent without encryption.

$R_{i,j}$ indicates how many PII types have been leaked and how severe they are. Its value is 1 if the most severe set of observed PII types have been leaked. $D_{i,j}$ indicates how much the APK is communicating with third-party domains. Its value is 1 if all the flows of the APK that generates the most flows are sent to third parties. Finally, $P_{i,j}$ indicates the amount of unencrypted traffic. Its value is 1 when all the traffic is sent over unencrypted connections.

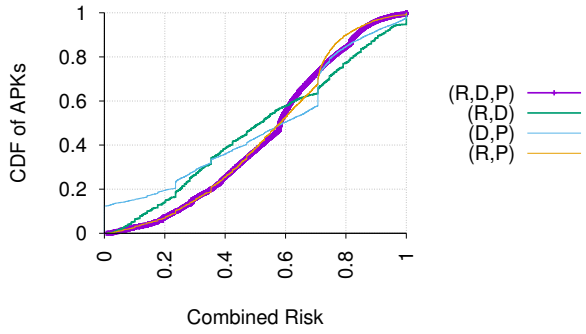


(a) $R_{i,j}$ vs $D_{i,j}$ (b) $R_{i,j}$ vs $P_{i,j}$ (c) $D_{i,j}$ vs $P_{i,j}$

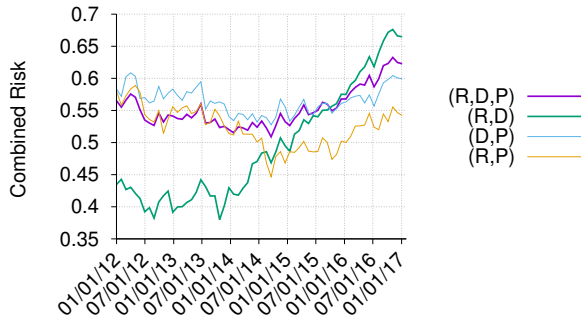
Fig. 6: **Two-dimensional risk analysis.** These plots are heat maps, where each cell represents the number of APKs $a_{i,j}$ in our dataset exhibiting the corresponding risk values x and y. Each axis represents one of the following privacy risks: *PII type risk* ($R_{i,j}$), *destination domain risk* ($D_{i,j}$), and *protocol risk* ($P_{i,j}$). Colors indicate the number of APKs with a given combined risk value, with red representing *five or more* APKs.

Notation	Explanation
$s(t) \in 0, \ldots, 5$	Privacy severity level for PII type t. $s(t)$:={ID=1; location=2; user-info=3; user-name=4; password=5}
$R_{i,j} \in [0,1]$	PII type risk for $a_{i,j}$, where τ is the set of types leaked and ν is the value corresponding to the most severe set of privacy leaks observed. $R_{i,j} = \frac{1}{\nu}\sum_{t\in\tau} 2^{s(t)-1}$
$D_{i,j} \in [0,1]$	Destination domain risk (third party vs first party) for $a_{i,j}$, where $h_{i,j}$ is the number flows generated by $a_{i,j}$, and $\rho_{i,j}$ is the number of flows in $h_{i,j}$ to third party domains. $D_{i,j} = \min\left(\frac{\rho_{i,j}}{\max_j h_{i,j}}, 1\right)$
$P_{i,j} \in [0,1]$	Protocol risk (plaintext vs encrypted) for $a_{i,j}$, where $\pi_{i,j}$ is the number of flows in $h_{i,j}$ that are in plaintext. $P_{i,j} = \frac{\pi_{i,j}}{h_{i,j}}$
$risk(x,y) \in [0,1]$ $risk(x,y,z) \in [0,1]$	Combined risk using normalized Euclidean distance. $risk(x,y) = \frac{1}{\sqrt{2}}\sqrt{x^2+y^2}$ $risk(x,y,z) = \frac{1}{\sqrt{3}}\sqrt{x^2+y^2+z^2}$

TABLE IX: **Definition of the privacy risk dimensions and risk combination metrics.**



(a) **Multidimensional combined risk by APK**. CDF of combined risk over all the APKs in our dataset. APKs are fairly evenly distributed across the risk spectrum.



(b) **Longitudinal variation of combined risk**. The x-axis represents the APK release date and the y-axis represents the combined $risk(...)$ metrics. Risk increases over time, and PII types and domains are by far the dominant factors for this trend.

Fig. 7: **Multidimensional privacy risk analysis.**

B. Combining Dimensions

We now combine the normalized risk metrics, choosing two or all three dimensions, and analyze how these combined privacy metrics change over time. We currently treat each dimension with equal weight, but note that different relative privacy concerns (e.g., PII leaks matter more than domains) can be captured by changing the relative weight of each dimension.

We begin by analyzing the two-dimensional combinations of privacy metrics, depicted using heatmaps in Figure 6. Each cell at (x,y) indicates the number of apps with risk scores of x and y, with red indicating five or more apps. Focusing on the combination of PII types leaked and destinations contacted (Figure 6a), we see several clusters emerge. The high density in the bottom left corner indicates that most APKs send relatively low-risk PII to relatively few domains. The points in the top left indicate that when high-risk PII is exposed by apps, they tend to leak it to few domains (with the exception of Pinterest, which contacts a large number of domains). Last, there are several apps that send moderately high-risk PII to many domains (right side of the figure).

When focusing on Figures 6b and 6c, we find that app behavior is fairly evenly spread across the x-axis range—indicating that there is no strong correlation between the fraction of TLS connections (x-axis) and privacy leaks (Fig. 6b) or number of domains contacted (Fig. 6c). The exception is that higher-risk PII tends to leak from apps using mostly encrypted

connections (top left), aside from a few cases near $x = 0.5$ (FastMeet, Meet24, Pinterest, Here WeGo - Offline Maps & GPS, ViewRanger Trails & Maps).

Based on the plots in Figure 6, we now define the *risk aggregation function*, which measures the normalized Euclidean distance between two different types of risk (see Table IX). This function captures the combination of different risks as a single number between 0 and 1.[15] Note that this function generalizes to arbitrary numbers of dimensions.

We first use the aggregate risk function to show in Figure 7a how all the possible combinations of the risk are distributed across all APKs in our dataset. The figure shows that most APKs are neither very low nor very high risk, and that the set of all APKs in our dataset are fairly evenly spread across the range of risk scores. Of course, because this does not consider time, it does not indicate whether recently released APKs are relatively higher or lower risk.

Is privacy getting better or worse? We investigate this question with Figure 7b, which shows a time series of the average privacy risk for APKs, grouped by release date. The figure shows a clear trend towards higher three-dimensional privacy risk over time (i.e., *risk*$(R_{i,j}, D_{i,j}, P_{i,j})$), with most of the increase attributable to the combination of more PII types being leaked and to more domains (the *risk*$(R_{i,j}, D_{i,j})$ curve). Thus, when it comes to leaking PII and contacting third parties, apps have gotten substantially worse over time.

To further analyze privacy risk changes, we conduct an app-focused analysis where we plot the combined risk score over time for each app (over all its APKs) and find the slope of the linear regression over these scores, as well as the standard deviation of the scores. Using this data, we categorize privacy risks per app as getting better, getting worse, staying similar, or exhibiting high variability over time. Algorithm 1 presents our classification logic when focusing on the combined score for R and D for each app. At a high level, we require that the slope and absolute difference between scores be sufficiently large to indicate that an app's privacy became worse or better. If the difference is not large and there is a relatively large standard deviation, then we indicate that the app is highly variable; otherwise, the app's privacy is labeled as similar.[16]

Using this approach, we calculated the following fractions of apps in each category: better (26.3%), worse (51.1%), similar (9.5%) and variable (13.1%). Thus, while a quarter of apps are getting better with respect to privacy, *twice as many are getting worse over time* and only a small fraction stay the same.

C. Privacy Risk Visualization

We built a web-based interactive tool [1] that allows individuals to explore the privacy risk data for any app in our dataset, showing how privacy risks changed across all versions of each app that the user selects. For this tool, we currently focus primarily on PII leak types, and allow the user to set relative leak severity for each PII category (denoted as $s(t)$ in Table IX); further, we compress our binary representation into a scale of 0 to 6 so that it is easier to understand for those who do not regularly think in terms of bit vectors. As part of ongoing work, we are investigating other intuitive ways to present our findings using a single score.

[15] Again, different scaling factors on each dimension can represent different relative risks between dimensions.

[16] The thresholds (θ_D, θ_S) were chosen heuristically, using 1.5 and 0.45 respectively. Users can explore other options via the web interface.

Algorithm 1 Trend Categorization for Privacy Risks.

```
function TREND(app)
    X ← list of versions
    Y ← list of normalized Euclidean distance of (R, D)
    Std ← Standard deviation of Y
    s ← Slope of the linear regression line of (X, Y)
    Y' ← s · X + intercept
    Df ← Y'_max − Y'_min
    Trend ← "similar"
    if Df ≥ θ_D then
        if s > 0 then Trend ← "worse"
        else Trend ← "better"
    else if Std > θ_S then Trend ← "variable"
    return Trend
```

VII. Conclusion

This paper provides the first longitudinal study of the privacy impact of using popular Android apps and their updated versions over time. We found that the PII shared with other parties changes over time, with the following trends: (1) overall privacy tends to worsen across versions; (2) the types of gathered PII change across versions, limiting the generalizability of single-version studies; (3) HTTPS adoption is relatively slow for mobile apps; (4) third parties not only track users pervasively, but also gather sufficient information to know what apps a user interacts with, when they do so, and where they are located when they do.

A naïve interpretation of our observed privacy trends is that users should stop updating apps; however, new versions of apps also contain bug fixes and improvements (e.g., critical security updates). Thus, what is needed is information that helps users make informed decisions when deciding whether to update the app given a set of changes in a new version. We envision that our online tool [1] can in part fill this need. Further, we recommend users to install tools like ReCon [50], Lumen [49], or AntMonitor [40] to block unwanted privacy leaks that come from newer versions of apps.

Our dataset and analysis code are available at: `https://recon.meddle.mobi/appversions/`.

Acknowledgments

We thank the anonymous reviewers for their helpful feedback. This work was partially supported by the Data Transparency Lab, the Academy of Finland PADS project (grant number 303815), the European Union under the H2020 TYPES (653449) project, and by DHS S&T contract FA8750-17-2-0145. This material is also based upon work supported by the NSF under Award No. CNS-1408632 and No. CNS-1564329, and a Security, Privacy and Anti-Abuse award from Google. Cloud computing resources were provided by an AWS Cloud Credits for Research award and by a Microsoft Azure for Research award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies or Google.

REFERENCES

[1] https://recon.meddle.mobi/appversions/.

[2] "AndroidAPKsFree," http://www.androidapksfree.com/.

[3] "APK4Fun," https://www.apk4fun.com/.

[4] "APKPure," https://apkpure.com/.

[5] "AppBrain," http://www.appbrain.com/.

[6] "geopy," https://github.com/geopy/geopy.

[7] "JustTrustMe," https://github.com/Fuzion24/JustTrustMe.

[8] "mitmproxy," https://mitmproxy.org/.

[9] Apple, "Supporting App Transport Security," https://developer.apple.com/news/?id=12212016b, December 2016.

[10] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps," in *Proc. of PLDI*, 2014.

[11] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing Android Permission Specification," in *Proc. of CCS*, 2012.

[12] M. Backes, S. Bugiel, and E. Derr, "Reliable Third-Party Library Detection in Android and its Security Applications," in *Proc. of CCS*, 2016.

[13] J. Ball, B. Schneier, and G. Greenwald, "NSA and GCHQ target Tor network that protects anonymity of web users," http://www.theguardian.com/world/2013/oct/04/nsa-gchq-attack-tor-network-encryption, October 2013.

[14] M. A. Bashir, S. Arshad, W. Robertson, and C. Wilson, "Tracing Information Flows Between Ad Exchanges Using Retargeted Ads," in *Proc. of USENIX Security*, 2016.

[15] T. Book, A. Pridgen, and D. S. Wallach, "Longitudinal Analysis of Android Ad Library Permissions," in *Proc. of MoST*, 2013.

[16] P. Carter, C. Mulliner, M. Lindorfer, W. Robertson, and E. Kirda, "CuriousDroid: Automated User Interface Interaction for Android Application Analysis Sandboxes," in *Proc. of FC*, 2016.

[17] T. Chen, I. Ullah, M. A. Kaafar, and R. Boreli, "Information Leakage through Mobile Analytics Services," in *Proc. of HotMobile*, 2014.

[18] S. R. Choudhary, A. Gorla, and A. Orso, "Automated Test Input Generation for Android: Are We There Yet?" in *Proc. of ASE*, 2015.

[19] S. Comino, F. M. Manenti, and F. Mariuzzo, "Updates Management in Mobile Applications. iTunes vs Google Play," in *SSRN*, 2016.

[20] K. Conger, "Apple will require HTTPS connections for iOS apps by the end of 2016," https://techcrunch.com/2016/06/14/apple-will-require-https-connections-for-ios-apps-by-the-end-of-2016, June 2016.

[21] A. Continella, Y. Fratantonio, M. Lindorfer, A. Puccetti, A. Zand, C. Kruegel, and G. Vigna, "Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis," in *Proc. of NDSS*, 2017.

[22] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "PiOS: Detecting Privacy Leaks in iOS Applications," in *Proc. of NDSS*, 2011.

[23] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *Proc. of USENIX OSDI*, 2010.

[24] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben, "Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security," in *Proc. of CCS*, 2012.

[25] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz, "Measuring HTTPS Adoption on the Web," in *Proc. of USENIX Security*, 2017.

[26] FTC, "Mobile Privacy Disclosures: Building Trust Through Transparency," *FTC Staff Report*, Feb 2013.

[27] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The Most Dangerous Code in the World: Validating SSL Certificates in Non-browser Software," in *Proc. of CCS*, 2012.

[28] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "RERAN: Timing- and Touch-sensitive Record and Replay for Android," in *Proc. of ICSE*, 2013.

[29] Google, "Android Developers Dashboards," https://developer.android.com/about/dashboards/index.html.

[30] ——, "App Manifest," https://developer.android.com/guide/topics/manifest/manifest-element.html.

[31] ——, "Google Maps Geocoding API," https://developers.google.com/maps/documentation/geocoding.

[32] ——, "Google Play Console Help: Advertising ID," https://support.google.com/googleplay/android-developer/answer/6048248.

[33] ——, "UI/Application Exerciser Monkey," https://developer.android.com/tools/help/monkey.html.

[34] ——, "Google Play Services 4.0," https://android-developers.googleblog.com/2013/10/google-play-services-40.html, October 2013.

[35] A. Hannak, P. Sapiezynski, A. Molavi Kakhki, B. Krishnamurthy, D. Lazer, A. Mislove, and C. Wilson, "Measuring Personalization of Web Search," in *Proc. of WWW*, 2013.

[36] A. Hannak, G. Soeller, D. Lazer, A. Mislove, and C. Wilson, "Measuring Price Discrimination and Steering on E-commerce Web Sites," in *Proc. of IMC*, 2014.

[37] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan, "PUMA: Programmable UI-automation for Large-scale Dynamic Analysis of Mobile Apps," in *Proc. of MobiSys*, 2014.

[38] C. Johnson, III, "US Office of Management and Budget Memorandum M-07-16," http://www.whitehouse.gov/sites/default/files/omb/memoranda/fy2007/m07-16.pdf, May 2007.

[39] A. Klyubin, "Protecting against unintentional regressions to cleartext traffic in your Android apps," https://security.googleblog.com/2016/04/protecting-against-unintentional.html, April 2016.

[40] A. Le, J. Varmarken, S. Langhoff, A. Shuba, M. Gjoka, and A. Markopoulou, "AntMonitor: A System for Monitoring from Mobile Devices," in *Proc. of Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data*, 2015.

[41] I. Leontiadis, C. Efstratiou, M. Picone, and C. Mascolo, "Don't kill my ads! Balancing Privacy in an Ad-Supported Mobile Application Market," in *Proc. of HotMobile*, 2012.

[42] C. Leung, J. Ren, D. Choffnes, and C. Wilson, "Should You Use the App for That? Comparing the Privacy Implications of App- and Web-based Online Services," in *Proc. of IMC*, 2016.

[43] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer, "Andrubis - 1,000,000 Apps Later: A View on Current Android Malware Behaviors," in *Proc. of BADGERS*, 2014.

[44] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An Input Generation System for Android Apps," in *Proc. of ESEC/FSE*, 2013.

[45] B. Naylor, "Congress Overturns Internet Privacy Regulation," http://www.npr.org/2017/03/28/521831393/congress-overturns-internet-privacy-regulation, March 2017.

[46] X. Pan, X. Wang, Y. Duan, X. Wang, and H. Yin, "Dark Hazard: Learning-based, Large-scale Discovery of Hidden Sensitive Operations in Android Apps," in *Proc. of NDSS*, 2017.

[47] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Rage Against the Virtual Machine: Hindering Dynamic Analysis of Android Malware," in *Proc. of EuroSec*, 2014.

[48] A. Razaghpanah, A. A. Niaki, N. Vallina-Rodriguez, S. Sundaresan, J. Amann, and P. Gill, "Studying TLS Usage in Android Apps," in *Proc. of CoNEXT*, 2017.

[49] 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.

[50] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. R. Choffnes, "ReCon: Revealing and Controlling Privacy Leaks in Mobile Network Traffic," in *Proc. of MobiSys*, 2016.

[51] S. Seneviratne, H. Kolamunna, and A. Seneviratne, "A Measurement Study of Tracking in Paid Mobile Applications," in *Proc. of WiSec*, 2015.

[52] Y. Song and U. Hengartner, "PrivacyGuard: A VPN-based Platform to Detect Information Leakage on Android Devices," in *Proc. of SPSM*, 2015.

[53] V. F. Taylor and I. Martinovic, "Short Paper: A Longitudinal Study of Financial Apps in the Google Play Store," in *Proc. of FC*, 2017.

[54] ——, "To Update or Not to Update: Insights From a Two-Year Study of Android App Evolution," in *Proc. of ASIACCS*, 2017.

[55] Y. Tian, B. Liu, W. Dai, B. Ur, P. Tague, and L. F. Cranor, "Supporting Privacy-Conscious App Update Decisions with User Reviews," in *Proc. of SPSM*, 2015.

[56] N. Vallina-Rodriguez, S. Sundaresan, A. Razaghpanah, R. Nithyanand, M. Allman, C. Kreibich, and P. Gill, "Tracking the Trackers: Towards Understanding the Mobile Advertising and Tracking Ecosystem," in *Proc. of the Workshop on Data and Algorithmic Transparency (DAT)*, 2016.

[57] T. Vidas and N. Christin, "Evading Android Runtime Analysis via Sandbox Detection," in *Proc. of ASIACCS*, 2014.

[58] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. Wang, "AppIntent: Analyzing Sensitive Data Transmission in Android for Privacy Leakage Detection," in *Proc. of CCS*, 2013.