Automated Analysis of Privacy Requirements for Mobile Apps

Sebastian Zimmeck*°, Ziqi Wang*, Lieyong Zou*, Roger Iyengar†°, Bin Liu*,
Florian Schaub‡°, Shomir Wilson§°, Norman Sadeh*, Steven M. Bellovin¶ and Joel Reidenberg‖
*School of Computer Science, Carnegie Mellon University, szimmeck@andrew.cmu.edu, sadeh@cs.cmu.edu
†Department of Computer Science and Engineering, Washington University in St. Louis
‡School of Information, University of Michigan
§Department of Electrical Engineering & Computing Systems, University of Cincinnati
¶Department of Computer Science, Columbia University
‖School of Law, Fordham University

***Abstract*—Mobile apps have to satisfy various privacy requirements. Notably, app publishers are often obligated to provide a privacy policy and notify users of their apps' privacy practices. But how can a user tell whether an app behaves as its policy promises? In this study we introduce a scalable system to help analyze and predict Android apps' compliance with privacy requirements. We discuss how we customized our system in a collaboration with the California Office of the Attorney General. Beyond its use by regulators and activists our system is also meant to assist app publishers and app store owners in their internal assessments of privacy requirement compliance.**

Our analysis of 17,991 free Android apps shows the viability of combining machine learning-based privacy policy analysis with static code analysis of apps. Results suggest that 71% of apps that lack a privacy policy should have one. Also, for 9,050 apps that have a policy, we find many instances of potential inconsistencies between what the app policy seems to state and what the code of the app appears to do. In particular, as many as 41% of these apps could be collecting location information and 17% could be sharing such with third parties without disclosing so in their policies. Overall, each app exhibits a mean of 1.83 potential privacy requirement inconsistencies.

I. Introduction

"We do not ask for, track, or access any location-specific information [...]." This is what Snapchat's privacy policy stated.[1] However, its Android app transmitted Wi-Fi- and cell-based location data from users' devices to analytics service providers. These discrepancies remained undetected before they eventually surfaced when a researcher examined Snapchat's data deletion mechanism. His report was picked up by the Electronic Privacy Information Center and brought to the attention of the Federal Trade Commission (FTC), which launched a formal investigation requiring Snapchat to implement a comprehensive privacy program.[2]

The case of Snapchat illustrates that mobile apps are often non-compliant with privacy requirements. However, any inconsistencies can have real consequences as they may lead to enforcement actions by the FTC and other regulators. This is especially true if discrepancies continue to exist for many years, which was the case for Yelp's collection of childrens' information.[3] These findings not only demonstrate that regulators could benefit from a system that helps them identify potential privacy requirement inconsistencies, but also that it would be a useful tool for companies in the software development process. This would be valuable because researchers found that privacy violations often appear to be based on developers' difficulties in understanding privacy requirements [7] rather than on malicious intentions. Thus, for example, tools that automatically detect and describe third-party data collection practices may be helpful for developers [7]. Consequently, it is a major motivation of our work to help companies identify red flags before they develop into serious and contentious privacy problems.

On various occasions, the FTC, which is responsible for regulating consumer privacy on the federal level, expressed dissatisfaction with the current state of apps' privacy compliance. Three times it manually surveyed childrens' apps [28], [29], [33] and concluded that the "results of the survey are disappointing" [29]. Deviating from mandatory provisions, many publishers did not disclose what types of data they collect, how they make use of the data, and with whom the data is shared [29]. A similar examination of 121 shopping apps revealed that many privacy policies are vague and fail to convey how apps actually handle consumers' data [32]. Given that the FTC limited its investigations to small samples of apps, a presumably large number of discrepancies between apps and their privacy policies remain undetected.

In this study we are presenting a privacy analysis system for Android that checks data practices of apps against

°Part of this work was conducted while Sebastian Zimmeck was a PhD student at Columbia University working in Prof. Sadeh's group at Carnegie Mellon University. Roger Iyengar, Florian Schaub, and Shomir Wilson were all in Prof. Sadeh's group at Carnegie Mellon University while involved in this research project, Roger Iyengar as an NSF REU undergraduate student, Florian Schaub as a post-doctoral fellow, and Shomir Wilson as a project scientist.

[1]Complaint In the Matter of Snapchat, Inc. (December 31, 2014).

[2]Decision and Order In the Matter of Snapchat, Inc. (December 31, 2014).

[3]United States of America v. Yelp, Inc. (September 17, 2014).

NDSS '17, 26 February - 1 March 2017, San Diego, CA, USA

http://dx.doi.org/10.14722/ndss.2017.23034

privacy requirements derived from their privacy policies and selected laws. Our work enables app publishers to identify potentially privacy-invasive practices in their apps before they are published. Moreover, our work can also aid governmental regulators, such as the FTC, to achieve a systematic enforcement of privacy laws on a large scale. App store owners, researchers, and privacy advocates alike might also derive value from our study. Our main contribution consists of the novel combination of machine learning (ML) and static analysis techniques to analyze apps' potential non-compliance with privacy requirements. However, we want to emphasize that we do not claim to resolve challenges in the individual techniques we leverage beyond what is necessary for our purposes. This holds especially true for the static analysis of mobile apps and its many unresolved problems, for example, in the analysis of obfuscated code. That said, the details of our contribution are as follows:

1) For a set of 17,991 Android apps we check whether they have a privacy policy. For the 9,295 apps that have one we apply machine learning classifiers to analyze policy content based on a human-annotated corpus of 115 policies. We show, for instance, that only 46% of the analyzed policies describe a notification process for policy changes. (§ III).
2) Leveraging static analysis we investigate the actual data practices occurring in the apps' code. With a failure rate of 0.4%, a mean F-1 score of 0.96, and a mean analysis time of 6.2 seconds per app our approach makes large-scale app analyses for legally relevant data practices feasible and reliable. (§ IV).
3) Mapping the policy to the app analysis results we identify and analyze potential privacy requirement inconsistencies between policies and apps. We also construct a statistical model that helps predict such potential inconsistencies based on app metadata. For instance, apps with a Top Developer badge have significantly lower odds for the existence of potential inconsistencies. (§ V).
4) In collaboration with the California Office of the Attorney General we performed a preliminary evaluation of our system for use in privacy enforcement activities. Results suggest that our system can indeed help their lawyers and other users to efficiently analyze salient privacy requirements allowing them to prioritize their work towards the most critical areas. (§ VI).

II. Related Work

We leverage prior work in privacy policy analysis (§ II-A), mobile app analysis (§ II-B), and their combination to identify potential privacy requirement inconsistencies (§ II-C).

A. Privacy Policy Analysis

Privacy policies disclose an organization's data practices. Despite efforts to make them machine-readable, for instance, via P3P [17], natural language policies are the de-facto standard. However, those policies are often long and difficult to read. Few lay users ever read them and regulators lack the resources to systematically review their contents. For instance, it took 26 data protection agencies one week, working together as the Global Privacy Enforcement Network (GPEN), to analyze the policies of 1,211 apps [38]. While various works aim to make privacy policies more comprehensible [34], there is a glaring absence of an automated system to accurately analyze policy content. In this study we aim for a solution. We want to automate and scale the analysis of natural language privacy policies. As of now, Massey et al. provided the most extensive evaluation of 2,061 policies, however, not focusing on their legal analysis but rather their readability and suitability for identifying privacy protections and vulnerabilities from a requirements engineering perspective [48]. In addition, Hoke et al. [40] studied the compliance of 75 policies with self-regulatory requirements, and Cranor et al. analyzed structured privacy notice forms of financial institutions identifying multiple instances of opt out practices that appear to be in violation of financial industry laws [16].

Different from previous studies we analyze policies at a large scale with a legal perspective and not limited to the financial industry. We analyze whether policies are available, as sometimes required by various laws, and examine their descriptions of data collection and sharing practices. For our analysis we rely on the flexibility of ML classifiers [72] and introduce a new approach for privacy policy feature selection. Our work is informed by the study of Costante et al., who presented a completeness classifier to determine which data practice categories are included in a privacy policy [15] and proposed rule-based techniques to extract data collection practices [14]. However, we go beyond these works in terms of both breadth and depth. We analyze a much larger policy corpus and we focus on legal questions that have not yet been automatically analyzed. Different from many existing works that focus on pre-processing of policies, e.g. by using topic modeling [13], [63] and sequence alignment [46], [55] to identify similar policy sections and paragraphs, we are interested in analyzing policy content.

Supervised ML techniques, as used in this study, require ground-truth. To support the development of these techniques crowdsourcing has been proposed as a viable approach for gathering rich annotations from unstructured privacy policies [59], [68]. While crowdsourcing poses challenges due to the policies' complexity [56], assigning annotation tasks to experts [72] and setting stringent agreement thresholds and evaluation criteria [68] can in fact lead to reliable policy annotations. However, as it is a recurring problem that privacy policy annotations grapple with low inter-annotator agreement [56], [72], we introduce a measure for analyzing their reliability based on the notion that high annotator disagreement does not principally inhibit the use of the annotations for ML purposes as long as the disagreement is not systematic.

B. Mobile App Analysis

Different from the closest related works [22], [62], our analysis of Android apps reflects the fundamental distinction between first and third party data practices. Both have to be analyzed independently as one may be allowed while the other may not. First and third parties have separate legal relationships to a user of an app. Among the third parties, ad and analytics libraries are of particular importance. Gibler et al. found that ad libraries were responsible for 65% of the identified data sharing with the top four accounting for 43% [35]. Similarly, Demetriou et al. [18] explored their potential reach and Grace et al. [39] their security and privacy

risks. They find that the most popular libraries have the biggest impact on sharing of user data, and, consequently, our analysis of sharing practices focuses on those as well. In fact, 75% of apps' location requests serve the purpose of sharing it with ad networks [44].

One of our contributions lies in the extension of various app analysis techniques to achieve a meaningful analysis of apps' potential non-compliance with privacy requirements derived from their privacy policies and selected laws. The core functionality of our app analyzer is built on Androguard [20], a static analysis tool. In order to identify the recipients of data we create a call graph [35], [66] and use PScout [6], which is comparable to Stowaway [26], to check whether an app has the required permissions for making a certain Android API call or allowing a library to make such. Our work takes further ideas from FlowDroid [4], which targeted the sharing of sensitive data, its refinement in DroidSafe [36], and the ded decompiler for Android Application Packages (APKs) [23]. However, neither of the previous works is intended for large-scale privacy requirement analysis.

Static analysis is ideally suited for large-scale analysis. However, as it was recently shown [1], [45], it also has its limitations. First, to the extent that the static analysis of Android apps is limited to Java it cannot reach code in other languages. In this regard, it was demonstrated that 37% of Android apps contain at least one method or activity that is executed natively (i.e., in C/C++) [1]. In addition to native code there is another obstacle that was shown to make the static analysis of Android apps challenging: code obfuscation. A non-negligible amount of apps and libraries are obfuscated at the package or code level to prevent reverse engineering [45]. Finally, static analysis cannot be used to identify indirect techniques, such as reflection, which often occurs in combination with native code [1]. We will discuss how these limitations affect our study in § IV-B.

C. Potential Privacy Requirement Inconsistencies

While mobile app analysis has received considerable attention, the analysis results are usually not placed into a legal context. However, we think that it is particularly insightful to inquire whether the apps' practices are consistent with the disclosures made in their privacy policies and selected requirements from other laws. The legal dimension is an important one that gives meaning to the app analysis results. For example, for apps that do not provide location services the transfer of location data may appear egregious. Yet, a transfer might be permissible under certain circumstances if adequately disclosed in a privacy policy. Only few efforts have attempted to combine code analysis of mobile apps with the analysis of privacy policies, terms of service, and selected requirements. Such analysis can identify discrepancies between what is stated in a legal document and what is actually practiced in reality. We are filling this void by identifying potential privacy requirement inconsistencies through connecting the analyses of apps, privacy policies, and privacy laws.

Various studies, e.g., [71], [70], demonstrated how to create privacy documentation or even privacy policies from program code. Other works focused on comparing program behavior with non-legal texts. For example, Huang et al. proposed AsDroid to identify contradictions between apps and user interface texts [41]. Kong et al. introduced a system to infer security and privacy related app behavior from user reviews [42]. Gorla et al. [37] used unsupervised anomaly detection techniques to analyze app store descriptions for outliers, and Watanabe et al. [66] used keyword-based binary classifiers to determine whether a resource that an app accesses (e.g., location) is mentioned in the app's description.

Different from most previous studies we analyze apps' behavior for potential non-compliance with privacy requirements derived from their privacy policies and selected laws. A step in this direction was provided by Bhoraskar et al., who found that 80% of ads displayed in apps targeted at children linked to pages that attempt to collect personal information in violation of the law [8]. The closest results to our study were presented by Enck et al. [22] and Slavin et al. [62]. In an analysis of 20 apps Enck et al. found a total of 24 potential privacy law violations caused by transmission of phone data, device identifiers, or location data [22]. Slavin et al. proposed a system to help software developers detect potential privacy policy violations [62]. Based on mappings of 76 policy phrases to Android API calls they discovered 341 such potential violations in 477 apps.

Our approach is inspired by TaintDroid [22] and similar to the studies of Slavin et al. [62] and Yu et al. [69]. However, we move beyond their contributions. First, our privacy requirements cover privacy questions previously not examined. Notably, we address whether an app needs a policy and analyze the policy's content (i.e., whether it describes how users are informed of policy changes and how they can access, edit, and delete data). Different from Slavin et al. we also analyze the collection and sharing of contact information. Second, TaintDroid, is not intended to have app store wide scale. Third, previous approaches do not neatly match to legal categories. They do not distinguish between first and third party practices [22], [62], do not take into account negative policy statements (i.e., statements that an app does *not* collect certain data, as, for example, in the Snapchat policy quoted in § I) [62], and base their analysis on a dichotomy of strong and weak violations [62] unknown to the law. Fourth, we introduce techniques that achieve a mean accuracy of 0.94 and a failure rate of 0.4%, which improve over the closest comparable results of 0.8 and 21% [62], respectively.

III. Privacy Policy Analysis

In this section we present our automated large-scale ML analysis of privacy policies. We discuss the law on privacy notice and choice (§ III-A), our evaluation of how many apps have a privacy policy (§ III-B), and the analysis of policy content (§ III-C).

A. Privacy Notice and Choice

The privacy requirements analyzed here are derived from selected laws and apps' privacy policies. If a policy or app does not appear to adhere to a privacy requirement, we define a potential privacy requirement inconsistency to occur (which we also refer to as potential inconsistency or non-compliance). In this regard, we caution that a potential inconsistency does not necessarily mean that a law is violated. First, not all privacy

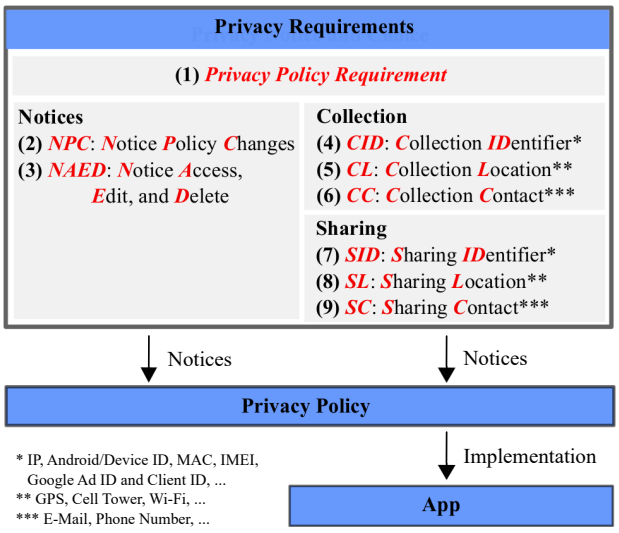

Fig. 1: *Per our privacy requirements, apps that process Personally Identifiable Information (PII) need to (1) have a privacy policy, (2-3) include notices about policy changes and access, edit, and deletion rights in their policy, (4-6) notify users of data collection practices, and (7-9) disclose how data is shared with third parties. The notice requirements for policy changes and access, edit, and deletion are satisfied by including the notices in the policies while the collection and sharing practices must be also implemented in the apps. We consider collection and sharing implemented in an app if it sends data to a first or third party server, respectively. Thus, it is not enough if data is kept on the phone but never sent.*

requirements might be applicable to all apps and policies. Second, our system is based on a particular interpretation of the law. While we believe that our interpretation is sound and in line with the enforcement actions of the FTC and other regulatory agencies, reasonable minds may differ.[4] Third, our system is based on machine learning and static analysis and, thus, by its very nature errors can occur. Figure 1 provides an overview of the law on notice and choice and the nine privacy requirements that our system analyzes (Privacy Policy Requirement, NPC, NAED, CID, CL, CC, SID, SL, SC).

As to the privacy policy requirement, there is no generally applicable federal statute demanding privacy policies for apps. However, California and Delaware enacted comprehensive online privacy legislation that effectively serves as a national minimum privacy threshold given that app publishers usually do not provide state-specific app versions or exclude California or Delaware residents. In this regard, the California Online Privacy Protection Act of 2003 (CalOPPA) requires online services that collect PII to post a policy.[5] The same is true according to Delaware's Online Privacy and Protection Act (DOPPA).[6] In addition, the FTC's Fair Information Practice Principles (FTC FIPPs) call for consumers to be given notice of an entity's information practices before any PII is collected [27]. Further, the Children's Online Privacy Protection Act of 1998 (COPPA) makes policies mandatory for apps directed to or known to be used by children.[7] Thus, we treat the existence of a privacy policy as a privacy requirement.

CalOPPA and DOPPA demand that privacy policies describe the process by which users are notified of policy changes.[8] COPPA also requires description of access, edit, and deletion rights.[9] Under the FTC FIPPs [27] as well as CalOPPA and DOPPA those rights are optional.[10] We concentrate our analysis on a subset of data types that are, depending on the context, legally protected: device IDs, location data, and contact information. App publishers are required to disclose the collection of device IDs (even when hashed) and location data.[11] Device IDs and location data are also covered by CalOPPA[12] and for childrens' apps according to COPPA.[13] The sharing of these types of information with third parties requires consent as well.[14] Our definition of sharing covers the direct data collection by third parties from first party apps.[15] Beyond device IDs and location data, contact information, such as e-mail addresses, may be protected, too.[16]

It should be noted that we interpret ad identifiers to be PII since they can be used to track users over time and across devices. We are also assuming that a user did not opt out of ads (because otherwise no ad identifiers would be sent to opted out ad networks). We further interpret location data to particularly cover GPS, cell tower, and Wi-Fi locations. We assume applicability of the discussed laws and perform our analysis based on the guidance provided by the FTC and the California Office of the Attorney General (Cal AG) in enforcement actions and recommendations for best practices (e.g., [27] and [11]). Specifically, we interpret the FTC actions as disallowing the omission of data practices in policies and assume that silence on a practice means that it does not occur.[17] Finally, we assume that all apps in the US Play store are subject to CalOPPA and DOPPA.[18] We believe this assumption is reasonable as we are not aware of any US app publisher excluding California or Delaware residents from app use or providing state-specific app versions.

B. Privacy Policy Requirement

To assess whether apps fulfill the requirement of having a privacy policy we crawled the Google Play store and downloaded a sample ($n = 17,991$) of free apps (full app set).[19] We started our crawl with the most popular apps and followed random links on their Play store pages to other apps. We included all categories in our crawl, however, excluded Google's Designed for Families program (as Google already

[4] We are focusing on the US legal system as we are most familiar with it. However, in principle, our techniques are applicable to any country with a privacy notice and choice regime.

[5] Cal. Bus. & Prof. Code §22575(a).

[6] Del. Code Tit. 6 §1205C(a).

[7] 16 CFR §312.4(d).

[8] Cal. Bus. & Prof. Code §22575(b)(3), Del. Code Tit. 6 §1205C(b)(3).

[9] 16 CFR §312.4(d)(3).

[10] Cal. Bus. & Prof. Code §22575(b)(2), Del. Code Tit. 6 §1205C(a).

[11] In the Matter of Nomi Technologies, Inc. (September 3, 2015).

[12] Cal. Bus. & Prof. Code §22577(a)(6) and (7) [11].

[13] 16 CFR §312.2(7) and (9).

[14] Complaint In the Matter of Goldenshores Technologies, LLC, and Erik M. Geidl (April 9, 2014).

[15] Cal. Bus. & Prof. Code §22575(b)(6), Del. Code Tit. 6 §1205C(b)(6).

[16] Complaint In the Matter of Snapchat, Inc. (December 31, 2014).

[17] Complaint In the Matter of Snapchat, Inc. (December 31, 2014).

[18] Cal. Bus. & Prof. Code §§22575–22579, Del. Code Tit. 6 §1205C.

[19] Whenever we refer to the Google Play store we mean its US site. Details on the various app and policy sets that we are using are described in Appendix A.

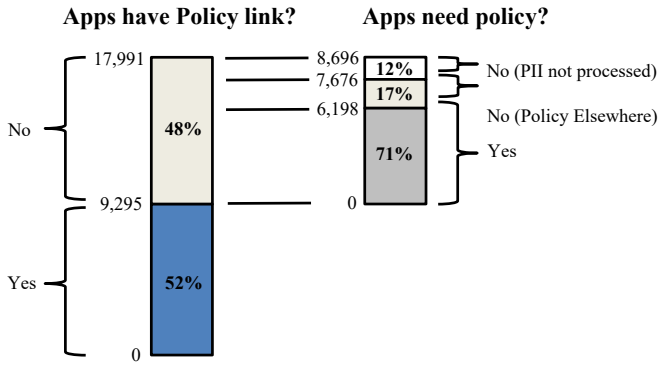

Fig. 2: *We analyze 17,991 free apps, of which 9,295 (52%) link to their privacy policy from the Play store (left). Out of the remaining apps, 6,198 (71%) appear to lack a policy while engaging in at least one data practice (i.e., PII is processed) that would require them to have one (right).*

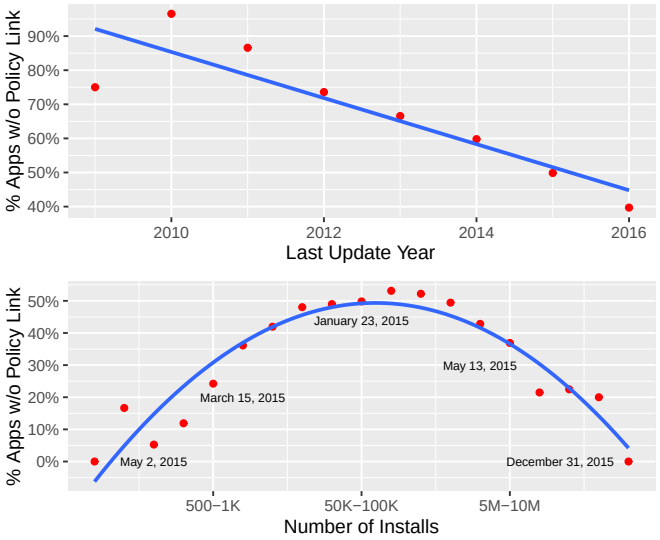

Fig. 3: *A linear regression model with the last app update year as independent variable and the percentage of apps without a policy link as dependent variable gives $r^2 = 0.79$ (top). In addition, a polynomial regression model using the number of installs as independent variable results in a multiple $r^2 = 0.9$ (bottom).*

requires apps in this program to have a policy) and Android Wear (as we want to focus on mobile apps). We assume that our sample is representative in terms of app categories, which we confirmed with a two-sample Kolmogorov-Smirnov goodness of fit test (two-tailed) against a sample of a million apps [49]. We could not reject the null hypothesis that both were drawn from the same distribution (i.e., $p > 0.05$). However, while the Play store hosts a long tail of apps that have fewer than 1K installs (56%) [49], we focus on more popular apps as our sample includes only 3% of such apps.

Potential Privacy Policy Requirement Inconsistencies. Out of all policies in the full app set we found that $n = 9,295$ apps provided a link to their policy from the Play store (full policy set) and $n = 8,696$ apps lacked such. As shown in Figure 2, our results suggest that 71% (6,198/8,696) apps without a policy link are indeed not adhering to the policy requirement. We used the Play store privacy policy links as proxies for actual policies, which we find reasonable since regulators requested app publishers to post such links [30], [11] and app store owners obligated themselves to provide the necessary functionality [10]. The apps in the full app set were offered by a total of 10,989 publishers, and their app store pages linked to 6,479 unique privacy policies.

We arrive at 71% after making two adjustments. First, if an app does not have a policy it is not necessarily non-compliant with the policy requirement. After all, apps that are not processing PII are not obligated to have a policy. Indeed, we found that 12% (1,020/8,696) of apps without a policy link are not processing PII and, thus, accounted for those apps. Second, despite the regulators' requests to post policy links in the Play store, some app publishers may still decide to post their policy elsewhere (e.g., inside their app). To account for that possibility we randomly selected 40 apps from our full app set that did not have a policy link in the Play store but processed PII. We found that 83% (33/40) do not seem to have a policy posted anywhere (with a Clopper-Pearson confidence interval (CI) ranging from 67% to 93% at the 95% level based on a two-tailed binomial test).[20] Thus, accounting for an additional 17% (1,478/8,696) of apps having a policy elsewhere leaves us with $100\% - 12\% - 17\% = 71\%$ out of $n = 8,696$ apps to be potentially non-compliant with the policy requirement.

Predicting Potential Privacy Policy Requirement Inconsistencies. As it appears that apps with frequent updates typically have a policy, we evaluate this hypothesis on our full app set using Pearson's Chi square test of independence. Specifically, it is our null hypothesis that whether an app has a policy is independent from the year when it was most recently updated. As the test returns $p \leq 0.05$, we can reject the null hypothesis at the 95% confidence level. Indeed, as shown in the linear regression model of Figure 3 (top), apps with recent update years have more often a policy than those that were updated longer ago. In addition to an app's update year there are other viable predictors. As shown in the polynomial regression model of Figure 3 (bottom) the number of installs is insightful. Apps with high install rates have more often a policy than apps with average install rates ($p \leq 0.05$). Surprisingly, the same is also true for apps with low install rates. An explanation could be that those are more recent apps that did not yet gain popularity. Indeed, apps with low install rates are on average more recently updated than apps with medium rates. For example, apps with 500 to 1K installs were on average updated on March 15, 2015 while apps with 50K to 100K installs have an average update date as of January 23, 2015.

Further, apps with an Editors' Choice or Top Developer badge usually have a policy, which is also true for apps that offer in-app purchases. It is further encouraging that apps with a content rating for younger audiences often have a policy.

[20] All CIs in this paper are based on a two-tailed binomial test and the Clopper-Pearson interval at the 95% level.

Practice	*No. Ann*	Ag_{pol}	*% Ag_{pol}*	*$Fleiss_{pol}$/$Krip_{pol}$*
NPC	395	86/115	75%	0.64
NAED	414	80/115	70%	0.59
CID	449	**92/115**	80%	**0.72**
CL	326	85/115	74%	0.64
CC	830	86/115	75%	0.5
SID	90	101/115	88%	**0.76**
SL	51	95/115	83%	0.48
SC	276	85/115	74%	0.58

TABLE I: *Absolute numbers of annotations (No. Ann) and various agreement measures, specifically, absolute agreements (Ag_{pol}), percentage agreements (% Ag_{pol}), Fleiss' κ ($Fleiss_{pol}$), and Krippendorff's α ($Krip_{pol}$). All agreement measures are computed on the full corpus of 115 policies and on a per-policy basis (e.g., for 92 out of 115 policies the annotators agreed on whether the policy allows collection of identifiers).*

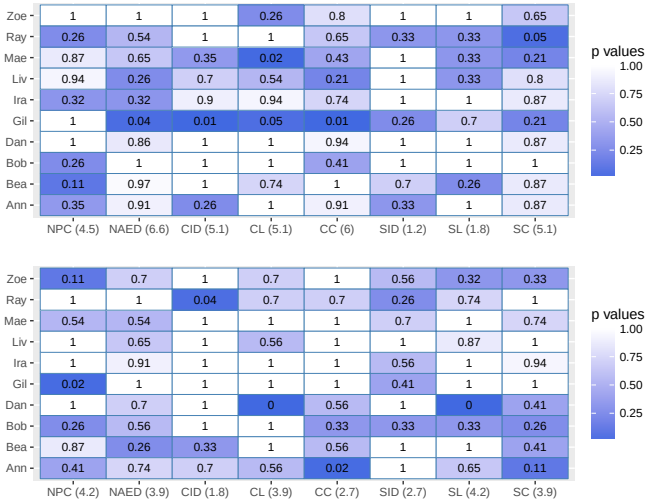

Fig. 4: *Analysis of systematic disagreement among annotators for the different data practices with binomial tests. Larger p values mean fewer disagreements. If there are no disagreements, we define $p = 1$. An annotator can be in the minority when omitting an annotation that the two other annotators made (top) or adding an extra annotation (bottom). Our results show few instances of systematic disagreement. The numbers in parentheses show the average numbers of absolute disagreements per annotator for the respective practices.*

Most apps for Everyone 10+ (75%), Teen (65%), and Mature 17+ (66%) audiences have a policy while apps that have an Everyone rating (52%) or are unrated (30%) often lack one.[21] Further, various app categories are particularly susceptible for not having a policy. Apps in the Comics (20%), Libraries & Demo (10%), Media & Video (28%), and Personalization (28%) categories have particularly low policy percentages, as compared to an average of 52% of apps having a policy across categories. Combining these predictors enables us to zoom in to areas of apps that are unlikely to have a policy. For instance, in the Media & Video category the percentage of apps with a policy decreases from 28% for rated apps to 12% for unrated apps. A similar decrease occurs in the Libraries & Demo category from 10% to 8%.

C. Privacy Policy Content

We now move from examining whether an app has a policy to the analysis of policy content (i.e., privacy requirements 2-9 in Figure 1). As a basis for our evaluation we use manually created policy annotations.

1) Inter-annotator Agreement: For training and testing of our policy classifiers we leverage the OPP-115 corpus [67]—a corpus of 115 privacy policies annotated by ten law students that includes 2,831 annotations for the practices discussed in this study. The annotations, which are described in detail in [67], serve as the ground-truth for evaluating our classifiers. Each annotator annotated a mean of 34.5 policies (median 35). We select annotations according to majority agreement (i.e., two out of three annotators agreed on it). As it is irrelevant from a legal perspective how often a practice is described in a policy, we measure whether annotators agree that a policy describes a given practice at least once.

High inter-annotator agreement signals the reliability of the ground-truth on which classifiers can be trained and tested. As agreement measures we use Fleiss' κ and Krippendorff's α, which indicate that agreement is good above 0.8, fair between 0.67 and 0.8, and doubtful below 0.67 [47]. From our results in Table I it follows that the inter-annotator agreement for collection and sharing of device IDs with respective values of 0.72 and 0.76 is fair. However, it is below 0.67 for the remaining classes. While we would have hoped for stronger agreement, the annotations with the observed agreement levels can still provide reliable ground-truth as long as the classifiers are not misled by patterns of systematic disagreement, which can be explored by analyzing the disagreeing annotations [57].

To analyze whether disagreements contain systematic patterns we evaluate how often each annotator disagrees with the other two annotators. If he or she is in a minority position for a statistically significant number of times, there might be a misunderstanding of the annotation task or other systematic reason for disagreement. However, if there is no systematic disagreement, annotations are reliable despite low agreement levels [57]. Assuming a uniform distribution each annotator should be in the minority in 1/3 of all disagreements. We test this assumption with the binomial test for goodness of fit. Specifically, we use the binomial distribution to calculate the probability of an annotator being x or more times in the minority by adding up the probability of being exactly x times in the minority, being $x+1$ times in the minority, up to $x+n$ (that is, being always in the minority), and comparing the result to the expected probability of 1/3. We use a one-tailed test as we are not interested in finding whether an annotator is fewer times in the minority than in 1/3 of the disagreements.

As shown in Figure 4, we only found few cases with systematic disagreement. More specifically, for 7% (11/160) of disagreements we found statistical significance ($p \leq 0.05$) for rejecting the null hypothesis at the 95% confidence level that the disagreements are equally distributed. We see that nearly half of the systematic disagreements occur for Gil. However, excluding Gil's and other affected annotations from the training

[21] Ratings follow the Entertainment Software Rating Board (ESRB) [24].

Practice	*Classifier*	*Parameters*	*Base* (n=40)	*Acc$_{pol}$* (n=40)	*95% CI* (n=40)	*Prec$_{neg}$* (n=40)	*Rec$_{neg}$* (n=40)	*F-1$_{neg}$* (n=40)	*F-1$_{pos}$* (n=40)	*Pos* (n=9,050)
NPC	SVM	RBF kernel, weight	0.7	0.9	0.76–0.97	0.79	0.92	**0.85**	0.93	**46%**
NAED	SVM	linear kernel	0.58	0.75	0.59–0.87	0.71	0.71	**0.71**	0.78	**36%**
CID	Log. Reg.	LIBLINEAR solver	0.65	0.83	0.67–0.93	0.77	0.71	**0.74**	0.87	46%
CL	SVM	linear kernel	0.53	0.88	0.73–0.96	0.83	0.95	**0.89**	0.86	34%
CC	Log. Reg.	LIBLINEAR, L2, weight	0.8	0.88	0.73–0.96	0.71	0.63	**0.67**	0.92	56%
SID	Log. Reg.	LBFGS solver, L2	0.88	0.88	0.73–0.96	0.94	0.91	**0.93**	**0.55**	**10%**
SL	SVM	linear kernel, weight	0.95	0.93	0.8–0.98	0.97	0.95	**0.96**	-	**12%**
SC	SVM	poly kernel (4 degrees)	0.73	0.78	0.62–0.89	0.79	0.93	**0.86**	**0.47**	**6%**

TABLE II: *Classifiers, parameters, and classification results for the policy test set (n=40) and the occurrence of positive classifications (Pos) in a set of n=9,050 policies (full app/policy set). We obtained the best results by always setting the regularization constant to $C = 1$ and for NPC, CC, and SL adjusting weights inversely proportional to class frequencies with scikit-learn's `class_weight` (weight). Except for the SL practice, all classifiers' accuracies (Acc$_{pol}$) reached or exceeded the baseline (Base) of always selecting the most often occurring class in the training set. $Prec_{neg}$, Rec_{neg}, and F-1$_{neg}$ are the scores for the negative classes (e.g., data is not collected or shared) while F-1$_{pos}$ is the F-1 score for positive classes.*

```
def location_feature_extraction(policy):

 data_type_keywords = ['geo', 'gps']
 action_keywords = ['share', 'partner']
 relevant_sentences = ''
 feature_vector = ''

 for sentence in policy:
  for keyword in data_type_keywords:
   if (keyword in sentence):
    relevant_sentences += sentence

 words = tokenize(relevant_sentences)
 bigrams = ngrams(words,2)

 for bigram in bigrams:
  for keyword in action_keywords:
   if (keyword in bigram):
    feature_vector += bigram, bigram[0],
        bigram[1]

 return feature_vector
```

Listing 1: *Pseudocode for the location sharing practice.*

set for our classifiers had only little noticeable effect. For some practices the classification accuracy slightly increased, for others it slightly decreased. Thus, we believe that our annotations are sufficiently reliable to serve as ground-truth for our classifiers. As other works have already explored, low levels of agreement in policy annotations are common and do not necessarily reflect their unreliability [56], [72]. In fact, different from our approach here, it could be argued that an annotator's addition or omission of an annotation is not a disagreement with the others' annotations to begin with.

2) Feature Selection: One of the most important tasks for correctly classifying data practices described in privacy policies is appropriate feature selection. Listing 1 shows a simplified example of our algorithm for the location sharing practice. Using information gain and tf-idf we identified the most meaningful keywords for each practice and created sets of keywords. One set of keywords refers to the data type of the practices (e.g., for the location sharing practice `geo` and `gps`) and is used to extract all sentences from a policy that contain at least one of the keywords. On these extracted sentences we are using a second set of keywords that refers to the actions of a data practice (e.g., for the location sharing practice `share` and `partner`) to create unigram and bigram feature vectors [72]. Thus, for example, if the keyword "share" is encountered, the bigrams "not share" or "will share" would be extracted if the words before "share" are "not" and "will," respectively. The feature vectors created from bigrams (and unigrams) are then used to classify the practices. If no keywords are extracted, the classifier will select the negative class. We applied the Porter stemmer to all processed text.

For finding the most meaningful features as well as for the subsequent classifier tuning we performed nested cross-validation with 75 policies separated into ten folds in the inner loop and 40 randomly selected policies as held out test set (policy test set). We used the inner cross-validation to select the optimal parameters during the classifier tuning phase and the held out policy test set for the final measure of classification performance. We stratified the inner cross-validation to avoid misclassifications due to skewed classes. After evaluating the performance of our classifiers with the policy test set we added the test data to the training data for the final classifiers to be used in our large-scale analysis.

3) Classification: During the tuning phase we prototyped various classifiers with scikit-learn [51], a Python library. Support vector machines and logistic regression had the best performance. We selected classification parameters individually for each data practice.

Classifier Performance for Policy Test Set. The classification results for our policy test set, shown in Table II, suggest that the ML analysis of privacy policies is generally feasible. For the negative classifications our classifiers achieve *F-1$_{neg}$* scores between 0.67 and 0.96. These scores are the most important measures for our task because the identification of a potential privacy requirement inconsistency demands that a practice occurring in an app is *not* covered by its policy (§ V-A). Consequently, it is less problematic that the sharing practices, which are skewed towards the negative classes, have relatively low *F-1$_{pos}$* scores of 0.55 (SID) and 0.47 (SC) or could not be calculated (SL) due to a lack of true positives in the policy test set.

Classification Results for Full App/Policy Set. We applied our classifiers to the policies in the full app/policy set with

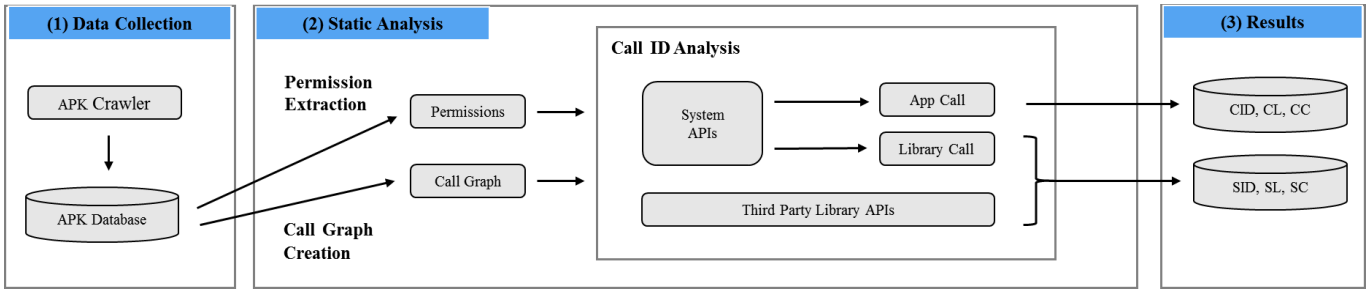

Fig. 5: *(1) Our system first crawls the US Google Play store for free apps. (2) Then, it performs for each app a static analysis. Specifically, it applies permission extraction, call graph creation, and call ID analysis, the latter of which is based on Android system and third party APIs. (3) Finally, results for the collection and sharing practices are generated and stored.*

$n = 9,050$ policies. We obtained this set by adjusting our full policy set ($n = 9,295$) to account for the fact that not every policy link might actually lead to a policy: for 40 randomly selected apps from our full policy set we checked whether the policy link in fact lead to a policy, which was the case for 97.5% (39/40) of links (with a CI of 0.87 to 1 at the 95% level). As the other 2.5%, that is, one link, lead to some other page and would not contain any data practice descriptions, we randomly excluded from the full policy set 2.5% = 245 of policies without any data practice descriptions leaving us with $n = 9,295 - 245 = 9,050$ policies in the full app/policy set. We emphasize that this technique does not allow us to determine whether the 245 documents actually did not contain a policy or had a policy that did not describe any practices. However, in any case the adjustment increases the occurrence of positive data practice instances in the full app/policy set keeping discrepancies between apps and policies at a conservative level as some apps for which the analysis did not find any data practice descriptions are now excluded.[22]

It appears that many privacy policies fail to satisfy privacy requirements. Most notably, per Table II, only 46% describe the notification process for policy changes, a mandatory requirement for apps under California and Delaware law. Similarly, only 36% of policies contain a statement on user access, edit, and deletion rights, which COPPA requires for childrens' apps, that is, apps intended for children or known to be used by children. For the sharing practices we expected more policies to engage in the SID, SL, and SC practices. The respective 10%, 12%, and 6% are rather small percentages for a presumably widely occurring practice, especially, given our focus on policies of free apps that often rely on targeted advertising.

Runtime Performance and Failure Rate. The analysis of all practices for the policies in the full app/policy set required about half an hour in total running ten threads in parallel on an Amazon Web Services (AWS) EC2 instance m4.4xlarge with 2.4 GHz Intel Xeon E5-2676 v3 (Haswell), 16 vCPU, and 64 GiB memory [2]. The feature extraction took up the majority of time and the training and classification finished in about one minute. There was no failure in extracting policy features or analyzing policies.

[22]We also checked the random sample of 40 apps for policies dynamically loaded via JavaScript because for such policies the feature extraction would fail. We had observed such dynamic loading before. However, as neither of the policies in the sample was loaded dynamically, we do not make an adjustment in this regard.

IV. Mobile App Analysis

In order to compare our policy analysis results to what apps actually appear to do we now discuss our app analysis approach. We begin with our system design (§ IV-A) and follow up with the system's analysis results (§ IV-B).

A. App Analysis System Design

Our app analysis system is based on Androguard [20], an open source static analysis tool written in Python that provides extensible analytical functionality. Apart from the manual intervention in the construction and testing phase, our system's analysis is fully automated. Figure 5 shows a sketch of our system architecture. A brief example for sharing of device IDs will convey the basic program flow of our data-driven static analysis.

For each app our system builds an API invocation map, which is utilized as a partial call graph (call graph creation). To illustrate, for sharing of device IDs all calls to the `android.telephony.TelephonyManager.getDeviceId` API are included in the call graph because the caller can use it to request a device ID. All calls to this and other APIs for requesting a device ID are added to the graph and passed to the identification routine (call ID analysis), which checks the package names of the callers against the package names of third party libraries to detect sharing of data. We focus on a set of ten popular libraries, which are listed in Table III.[23] In order to make use of the `getDeviceId` API a library needs the `READ_PHONE_STATE` permission. Only if the analysis detects that the library has the required permission (permission extraction), the app is classified as sharing device IDs with third parties.[24] We identified relevant Android API calls for the types of information we are interested in and the permission each call requires by using PScout [6].

Our static analysis is informed by a manual evaluation of Android and third party APIs. Because sharing of data most often occurs through third party libraries [23], we can leverage the insight that the observation of data sharing for a given library allows extension of that result to all apps

[23]The limitation on a small set of libraries allows us to manually analyze the library functions freeing us from using resource-intensive data flow analysis techniques in our app analysis. However, in principle, it is possible to include more libraries.

[24]Android's permission model as of Android 6.0 does not distinguish between permissions for an app and permissions for the app's libraries, which, thus, can request all permissions of the app.

3rd Party Library
Crashlytics/Fabric
Crittercism/Aptel.
Flurry Analytics
Google Analytics
Umeng
AdMob*
InMobi*
MoPub*
MillennialMedia*
StartApp*

TABLE III: *Ad* and analytics libraries.*

Pract	***Base*** *(n=40)*	***Acc***$_{app}$ *(n=40)*	***95% CI*** *(n=40)*	***Prec***$_{pos}$ *(n=40)*	***Rec***$_{pos}$ *(n=40)*	***F-1***$_{pos}$ *(n=40)*	***F-1***$_{neg}$ *(n=40)*	***Pos***$_{w/\ pol}$ *(n=9,295)*	***Pos***$_{w/o\ pol}$ *(n=8,696)*
CID	0.8	0.9	0.76–0.97	0.89	1	**0.94**	0.67	95%	87%
CL	0.55	0.8	0.64–0.91	0.73	1	**0.85**	0.71	66%	49%
CC	0.78	1	0.91–1	1	1	**1**	1	**25%**	12%
SID	0.68	0.95	0.83–0.99	1	0.93	**0.96**	0.93	**71%**	**62%**
SL	0.93	1	0.91–1	1	1	**1**	1	**20%**	16%
SC	0.98	1	0.91–1	1	1	**1**	1	**2%**	0%

TABLE IV: *App analysis results for the app test set (n=40) and the percentages of practices' occurrences in the full app set (n=17,991). More specifically, $Pos\_w/\ pol$ and $Pos\_w/o\ pol$ are showing what percentage of apps engage in a given practice for the subset of apps in the full app set with a policy (n=9,295) and without a policy (n=8,696), respectively. We measure precision, recall, and F-1 score for the positive and negative classes with the $_{pos}$ and $_{neg}$ subscripts designating the respective scores.*

using the same library [35]. As the top libraries have the farthest reach [35] we focus on those. We used AppBrain [3] to identify the ten most popular libraries by app count that process device ID, location, or contact data. To the extent we were able to obtain them we also analyzed previous library versions dating back to 2011. After all, apps sometimes continue to use older library versions even after the library has been updated. For each library we opened a developer account, created a sample app, and observed the data flows from the developer perspective. For these apps as well as for a sample of Google Play store apps that implement the selected libraries we additionally observed their behavior from the outside by capturing and decrypting packets via a man-in-the-middle attack and a fake certificate [54]. We also analyzed library documentations. These exercises allowed us to evaluate which data types were sent out to which third parties.

B. App Analysis Results

Performance Results for App Test Set. Before exploring the analysis results for the full app set we discuss the performance of our app analysis on a set of 40 apps (app test set), which we selected randomly from the publishers in the policy test set (to obtain corresponding app/policy test pairs for our later performance analysis of potential privacy requirement inconsistencies in § V-A). To check whether the data practices in the test apps were correctly analyzed by our system we dynamically observed and decrypted the data flows from the test apps to first and third parties, performed a manual static analysis for each test app with Androguard [20], and studied the documentations of third party libraries. Thus, for example, we were able to infer from the proper implementation of a given library that data is shared as explained in the library's documentation. We did not measure performance based on micro-benchmarks, such as DroidBench [4], as those do not fully cover the data practices we are investigating.

In the context of potential inconsistencies (§ V-A) correctly identifying positive instances of apps' collection and sharing practices is more relevant than identifying negative instances because only practices that *are* occurring in an app need to be covered in a policy. Thus, the results for the data practices with rarely occurring positive test cases are especially noteworthy: CC, SL, and SC all reached *F-1*$_{pos} = 1$ indicating that our static analysis is able to identify positive practices even if they rarely occur. Further, the F-1$_{pos}$ scores, averaging to a mean of 0.96, show the overall reliability of our approach. For all practices the accuracy is also above the baseline of always selecting the test set class that occurs the most for a given practice. Overall, as shown in Table IV, our results demonstrate the general reliability of our analysis.

Data Practice Results for Full App Set. For all six data practices we find a mean of 2.79 occurring practices per app for apps with policies and 2.27 occurrences for apps without policies. As all practices need to be described in a policy per our privacy requirements (§ III-A), it is already clear that there are substantial amounts of potential inconsistencies between apps and policies simply due to missing policies. For example, the SID practice was detected in 62% of apps that did not have a policy (Table IV), which, consequently, appear to be potentially non-compliant with privacy requirements. Furthermore, for apps that had a policy only 10% disclosed the SID practice (Table II) while it occurred in 71% of apps (Table IV). Thus, 61% of those apps are potentially non-compliant in this regard. The only practices for which we cannot immediately infer the existence of potential inconsistencies are the CC and SC practices with policy disclosures of 56% and 6% and occurrences in apps of 25% and 2%, respectively. We can think of two reasons for this finding.

First, there could be a higher sensitivity among app publishers to notify users of practices related to contact data compared to practices that involve device identifiers and location data. Publishers may categorize contact data more often as PII. Second, different from device ID and location data, contact information is often provided by the user through the app interface bypassing the APIs that we consider for our static analysis (most notably, the `android.accounts.AccountManager.getAccounts` API). Thus, our result demonstrates that the analysis approach has to be custom-tailored to each data type and that the user interface should receive heightened attention in future works [62]. It also illustrates that our results only represent a lower bound, particularly, for the sharing practices (SID, SL, SC), which are limited to data sent to the ten publishers of the libraries in Table III.

Limitations. There are various limitations of our static analysis. At the outset our approach is generally subject to the same limitations that all static analysis techniques for Android are facing, most notably, the difficulties of analyzing native

Practice	*Acc* (n=40)	$Acc_{pol} \cdot Acc_{app}$ (n=40)	*95% CI* (n=40)	$Prec_{pos}$ (n=40)	Rec_{pos} (n=40)	$F\text{-}1_{pos}$ (n=40)	$F\text{-}1_{neg}$ (n=40)	*MCC* (n=40)	*TP, FP, TN, FN* (n=40)	*Inconsistent* (n=9,050)
CID	**0.95**	0.74	0.83–0.99	**0.75**	1	**0.86**	0.97	0.84	6, 2, 32, 0	**50%**
CL	**0.83**	0.7	0.67–0.93	**0.54**	1	**0.7**	0.88	0.65	8, 7, 25, 0	**41%**
CC	**1**	0.88	0.91–1	-	-	-	1	-	0, 0, 40, 0	**9%**
SID	**0.85**	0.84	0.7–0.94	0.93	0.74	**0.82**	0.87	0.71	14, 1, 20, 5	**63%**
SL	**1**	0.93	0.91–1	1	1	**1**	1	1	3, 0, 37, 0	**17%**
SC	**1**	0.78	0.91–1	1	1	**1**	1	1	1, 0, 39, 0	**2%**

TABLE V: *Results for identifying potential privacy requirement inconsistencies in the app/policy test set (n=40) and the percentage of such potential inconsistencies for all 9,050 app/policy pairs (Inconsistent). Assuming independence of policy and app accuracies, $Acc_{pol} \cdot Acc_{app}$, that is, the product of policy analysis accuracy (Table II) and app analysis accuracy (Table IV), indicates worse results than the directly measured accuracy. The Matthews correlation coefficient (MCC), which is insightful for evaluating classifiers in skewed classes, indicates a positive correlation between observed and predicted classes.*

code, obfuscated code, and indirect techniques (e.g., reflection). However, there are various considerations that ameliorate exposure of our approach to these challenges. First, if an app or a library uses native code, it cannot hide its access to Android System APIs [35]. In addition, the use of native code in ad libraries is minimal [45]. Indeed, we have rarely encountered native code in our analysis. Similarly, the need to interact with a variety of app developers effectively prohibits the use of indirect techniques [9]. However, code obfuscation in fact presents an obstacle. Our static analysis failed in 0.4% (64/18,055) of all cases due to obfuscation (i.e., an app's Dex file completely in bytecode). However, our failure rate improves over the closest comparable rate of 21% [62].

It is a further limitation of our approach that the identification of data practices occurs from the outside (e.g., server-side code is not considered). While this limitation is not a problem for companies' analysis of their own apps, which we see as a major application of our approach, it can become prevalent for regulators, for instance. In many cases decrypting HTTPS traffic via a man-in-the-middle attack and a fake certificate will shed some light. However, it appears that some publishers are applying encryption inside their app or library. In those cases, the analysis will need to rely on inferring the data practice in question indirectly. For example, it remains possible to check whether a library is properly implemented in an app according to the library's documentation, which lends evidence to the inference that the app makes use of the documented practices.

Also, our results for the sharing practices only refer to the ten third parties listed in Table III. The percentages for sharing of contacts, device IDs, or locations would almost certainly be higher if we would consider additional libraries. In addition, our definition of sharing data with a third party only encompasses sharing data with ad networks and analytics libraries. However, as it was shown that ad libraries are the recipients of data in 65% of all cases [35], we believe that this definition covers a substantial portion of sharing practices. It should be finally noted that our investigation does not include collection or sharing of data that occurs through user input, offline, or at app servers' backends. However, as our analysis already identifies a substantial percentage of potentially non-compliant apps, we think that there is value in our techniques even with these limitations.

Runtime Performance. In terms of runtime performance, using ten threads in parallel on an AWS EC2 instance m4.10xlarge with 2.4 GHz Intel Xeon E5-2676 v3 (Haswell), 40 vCPU, and 160 GiB memory [2] the analysis of all 17,991 APKs took about 31 hours. The mean runtime is 6.2 seconds per APK analysis.

V. Identifying Potential Inconsistencies

In this section we unite our policy (§ III) and app (§ IV) analyses. We explore to which extent apps are potentially non-compliant with privacy requirements (§ V-A) and show how app metadata can be used to zoom in on sets of apps that have a higher likelihood of non-compliance (§ V-B).

A. Potential Inconsistencies in Individual App/Policy Pairs

Potential Inconsistencies from a Developer's Perspective. As the results of a survey among app developers suggest a lack of understanding privacy-best practices [7], it could be that many of the potential inconsistencies we encountered are a consequence of this phenomenon as well. Especially, many developers struggle to understand what type of data third parties receive, and with limited time and resources even self-described privacy advocates and security experts grapple with implementing privacy and security protections [7]. In this regard, our analysis approach can provide developers with a valuable indicator for instances of potential non-compliance. For identifying potential inconsistencies positive app classes and negative policy classes are relevant. In other words, if a data practice does not occur in an app, it does not need policy coverage because there can be no inconsistency to begin with. Similarly, if a user is notified about a data practice in a policy, it is irrelevant whether the practice is implemented in the app or not. Either way, the app is covered by the policy. Based on these insights we analyze the performance of our approach.

Performance Results for App/Policy Test Set. To evaluate the performance of our system for correctly identifying potential privacy requirement inconsistencies we use a test set with corresponding app/policy pairs (app/policy test set). The set contains the 40 random apps from our app test set (§ IV-B) and their associated policies from our policy test set (§ III-C3). We associate an app and a policy if the app or its Play store page links to the policy or if the policy explicitly declares itself applicable to mobile apps. As only 23 policies satisfy this requirement some policies are associated with multiple apps. As shown in Table V, accuracy results range between 0.83 and 1 with a mean of 0.94. Although not fully comparable,

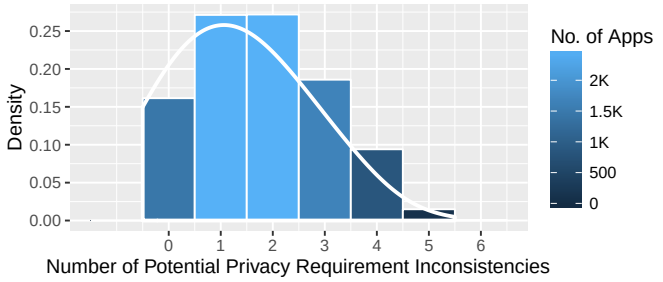

Fig. 6: *For the full app/policy set (n = 9,050) we found that 2,455 apps have one potential inconsistency, 2,460 have two, and only 1,461 adhere completely to their policy. Each app exhibits a mean of 1.83 (16,536/9,050) potential inconsistencies (with the following means per data practice: CID: 0.5, CL: 0.41, CC: 0.09, SID: 0.63, SL: 0.17, SC: 0.02).*

Variable	*Pos*	*p value*	*OR*	*95% CI*
No. User Ratings	100%	**0.0001**	$0.\overline{9}$	0.9999998–$0.\overline{9}$
Overall Score	**100%**	<**0.0001**	**1.4**	1.24–1.57
Badge	21%	<**0.0001**	0.57	0.49–0.65
In-app Purchases	27%	**0.08**	1.15	0.99–1.34
Interactive Elm	45%	<**0.0001**	1.33	1.17–1.53
Content Unrated	5%	**0.002**	0.68	0.53–0.87

TABLE VI: *Significant variables for predicting apps' potential non-compliance with at least one privacy requirement as evaluated on our full app/policy set (n=9,050). Top Developer and Editor's Choice badges are assigned by Google. Interactive elements and unrated content refer to the respective ESRB classifications [24]. Pos designates the percentages of positive cases (e.g., 100% apps have an overall score), OR is the odds ratio, and the 95% CI is the profile likelihood CI.*

AsDroid achieved an accuracy of 0.79 for detecting stealthy behavior [41] and Slavin et al. [62] report an accuracy of 0.8 for detecting discrepancies between app behavior and policy descriptions. For the 240 classification instances in the app/policy test set—that is, classifying six practices for each of the 40 app/policy pairs—our system correctly identified 32 potential inconsistencies (TP). It also returned five false negatives (FN), 10 false positives (FP), and 193 true negatives (TN).[25]

The F-1_{pos} scores for our analysis, ranging from 0.7 to 1, indicate the overall reliable identification of potential inconsistencies. While we think that these results are generally promising, we obtain a relatively low precision value of $Prec_{pos} = 0.54$ for the CL practice and for the CID practice we had hoped for a higher precision than $Prec_{pos} = 0.75$ as well. These results illustrate a broader point that is applicable beyond those practices. False positives seem to occur because our analysis takes into account too many APIs that are only occasionally used for purposes of the data practice in question. Despite our belief that it is better to err on the side of false positives, which is especially true for an auditing system [35], in hindsight we probably would have left out some APIs. The opposite problem seems to occur in the SID practice. We included too few relevant APIs. In this regard, we acknowledge the challenge of identifying a set of APIs that captures the bulk of cases for a practice without being over- or under-inclusive.

Potential Inconsistencies for Full App/Policy Set. As indicated by the high percentages shown in Table V, we identified potential inconsistencies on a widespread basis. Specifically, collection of device IDs and locations as well as sharing of device IDs are practices that are potentially inconsistent for 50%, 41%, and 63% of apps, respectively. However, given the relatively low precision and high recall for these practices, we caution that their percentages might be an overestimation. It is further noteworthy that for sharing of location and contact information nearly every detection of the practices goes hand in hand with a potential inconsistency. For the apps that share location information (20%, per Table IV) nearly all (17%, per Table V) do not properly disclose such sharing. Similarly, for the 2% of apps that share contact data only a handful provide sufficient disclosure.

[25]Appendix B describes details of calculating true and false positives and negatives.

The average number of 1.83 potential inconsistencies per app is high compared to the closest previous averages with 0.62 (113/182) cases of stealthy behavior [41] and potential privacy violations of 1.2 (24/20) [22] and 0.71 (341/477) [62]. Figure 6 shows details of our results. In this regard, it should be noted that for apps without a policy essentially every data collection or sharing practice causes a potential inconsistency. For example, all 62% apps without a policy that share device IDs (Table IV) are potentially non-compliant. Thus, overall our results suggest a broad level of potential inconsistency between apps and policies.[26]

B. Potential Inconsistencies for Groups of App/Policy Pairs

Analyzing individual apps for potential non-compliance at scale is a resource-intensive task. Thus, it is worthwhile to first estimate an app population's potential non-compliance as a whole before performing individual app analyses. Our suggestion is to systematically explore app metadata for correlations with potential inconsistencies based on statistical models. This broad macro analysis supplements the individual app analysis and reveals areas of concern on which, for example, privacy activists can focus on. To illustrate this idea we evaluate a binary logistic regression model that determines the dependence of whether an app has a potential inconsistency (the dependent variable) from six Play store app metadata variables (the independent variables). Our results, shown in Table VI, demonstrate correlations at various statistical significance levels with p values ranging from 0.0001 to 0.08. Particularly, with an increase in the number of user ratings the probability of potential inconsistencies decreases. There is also a decreasing effect for apps with a badge and for apps whose content has not yet been rated.

Interestingly, apps with higher overall Google Play store scores do not have lower odds for potential inconsistencies. In fact, the opposite is true. With an increase in the overall score the odds of a potential inconsistency become higher. An increase of the overall score by one unit, e.g., from 3.1 to 4.1 (on a scale of 1 through 5), increases the odds by a factor of 1.4. A reason could be that highly rated apps provide functionality and personalization based on user data,

[26]As we are evaluating our system for use in privacy enforcement activities (§ VI) we decided to abstain from contacting any affected app publishers of our findings.

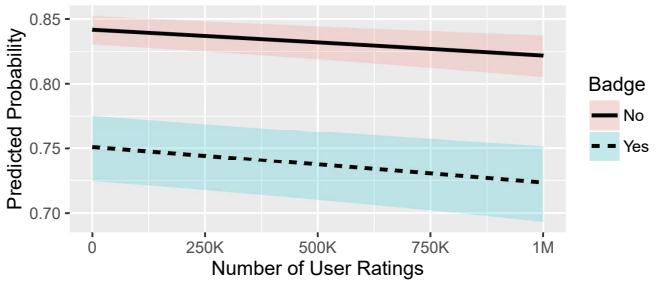

Fig. 7: *In our logistic model the predicted probability of an app having a potential inconsistency is dependent on the number of user ratings and assignment of a badge. The overall score is held at the mean and in-app purchases, interactive elements, and unrated content are held to be not present. The shaded areas identify the profile likelihood CIs at the 95% level.*

whose processing is insufficiently described in their privacy policies. Also, users do not seem to rate apps based on privacy considerations. We found the word "privacy" in only 1% (220/17,991) of all app reviews. Beyond an app's score, the odds for a potential inconsistency also increase for apps that feature in-app purchases or interactive elements. Also, supplementing our model with category information reveals that the odds significantly ($p \leq 0.05$) surge for apps in the Finance, Health & Fitness, Photography, and Travel & Local categories while they decrease for apps in the Libraries & Demo category.

In order to evaluate the overall model fit based on statistical significance we checked whether the model with independent variables (omitting the category variables) had significantly better fit than a null model (that is, a model with the intercept only). The result of a Chi square value of 151.03 with six degrees of freedom and value of $p \leq 0.001$ indicates that our model has indeed significantly better fit than the null model. To see the impact of selected aspects of the model it is useful to illustrate the predicted probabilities. An example is contained in Figure 7. Apps with a Top Developer or Editor's Choice badge have a nearly 10% lower probability of a potential inconsistency. That probability further decreases with more user ratings for both apps with and without badge.

VI. Case Study: Evaluating our System for Use by the Cal AG

We worked with the Cal AG, to evaluate our system's capabilities for supplementing the enforcement of CalOPPA [12]. To that end, we implemented a custom version of our system (§ VI-A) and added various new analysis features (§ VI-B). The feedback we received is encouraging and confirms that our system could enable regulators to achieve more systematic enforcement of privacy requirements (§ VI-C).[27]

A. System Implementation

As shown in Figure 8, we implemented our system for the Cal AG as a web application. It allows users to request analyses for individual apps and also supports batch processing. For

[27] We continue our work with the Cal AG beyond this study and expect further results.

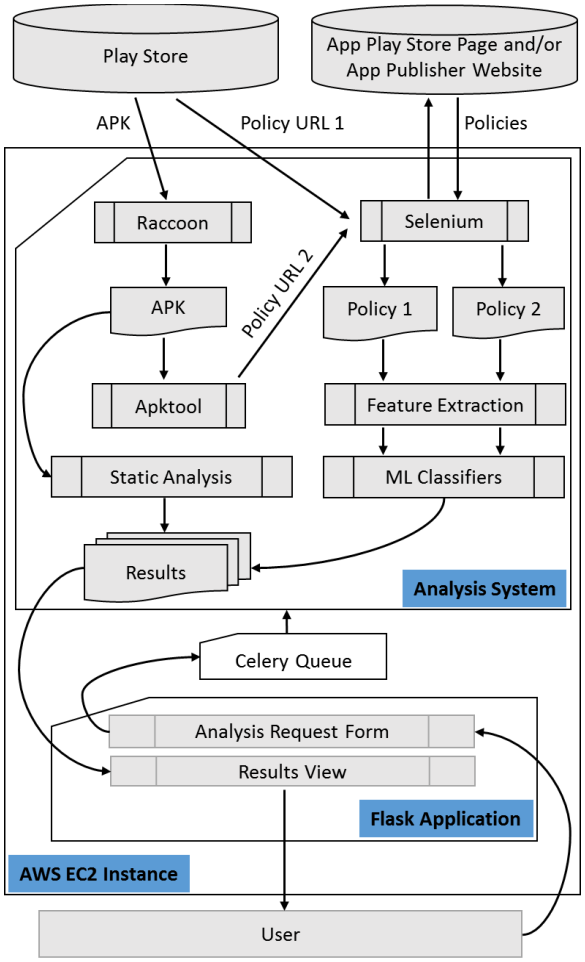

Fig. 8: *Our system allows users to analyze apps for potential privacy requirement non-compliance. An app can be subject to multiple privacy policies—for example, one policy from inside the app and one from the app's Play Store page. In these cases the app is checked against multiple policies.*

scalability reasons we chose to leverage AWS EC2 t2.large instances with up to 3.0 GHz Intel Xeon, 2 vCPU, and 8 GiB memory [2].

The system's graphical user interface applies the Flask Python web framework [58] running on an Apache web server [64] with Web Server Gateway Interface module [21]. All analysis requests are added to a Celery task queue [5], which communicates with the Flask application using the RabbitMQ message broker [52]. Once an analysis is finished the results are loaded by the Flask application and displayed in the users' browsers.

Similar as in our original system, all APKs are downloaded via Raccoon [50] and apps' privacy policy links are retrieved from their respective Play store pages. However, in order to download the websites that the links are leading to we automated a Firefox browser with Selenium [60] and PyVirtualDisplay [53]. Using a real browser instead of simply crawling the HTML of the privacy policy pages is advantageous as it can obtain policies that are loaded dynamically via JavaScript.

After the website with the privacy policy is downloaded any elements that are not part of the policy, such as page navigation elements, are removed. The system then runs our feature extraction routines (§ III-C2) as well as ML classifiers (§ III-C3) on the policy and the static analysis (§ IV) on the downloaded APK. Finally, the results are displayed to the user with flags raised for potential inconsistencies.

B. Additional Functionality

We placed high emphasis on usability from both a legal and human-computer interaction perspective. Notably, in some cases the Cal AG users were interested in receiving additional information. For instance, one additional piece of information was the breakdown of third parties in the sharing practices. The initial version of our system's report simply showed that the user's contact and device ID were shared, however, without disclosing that those were shared with, say, InMobi and Crashlytics. Distinguishing which type of information is shared with which third party is important under CalOPPA because the sharing of contact information makes a stronger case than the sharing of device IDs, for example.[28]

Given its importance we implemented additional contact information functionality. Because we believe that the relatively low detection rate for the collection and sharing of contact information (§ V-A) is due to the fact that such information is often manually entered by the user or obtained via other sources, we enhanced our system in this regard. Particularly, we leveraged the Facebook Login library that gives apps access to a user's name and Facebook ID [25]. The system detects the usage of the Facebook Login library in an app by extracting the app's manifest and resource files with Apktool [65] and then searching for signatures required for the Facebook Login to work properly. These include an activity or intent filter dedicated to the Login interface, a Login button in the layout, and the invocation of an initialization, destruction, or configuration routine.

Another feature that we added is the download of privacy policies linked to from within apps. Our initial policy crawler was limited to downloading policies via an app's Play store page. As the Cal AG provided guidance to app publishers for linking to the policy from both the Play store and from within an app [11], our new approach is intended to cover both possibilities. The system finds the links in an app by extracting strings from the APK file using Apktool and then extracting URLs from within these strings that contain relevant keywords, such as "privacy." If a policy link inside an app differs from the app's Play store policy link or if there are multiple policy links identified within an app, our system downloads and analyzes all documents retrieved from these links.

C. System Performance

The Cal AG users reported that our system has the potential to significantly increase their productivity. Particularly, as they have limited resources it can give them guidance on certain areas, e.g., categories of apps, to focus on. They can also put less effort and time into analyzing practices in apps for which our system does not find potential inconsistencies. Instead, they can concentrate on examining apps for which flags were raised. In addition, the Cal AG users expressed that our system was useful for estimating the current overall state of CalOPPA compliance. For example, the analysis results alerted them of some app policies that use vague language in the descriptions of their collection and sharing practices.

[28] Compare Cal. Bus. & Prof. Code §22577(a)(3), according to which an e-mail address by itself qualifies as PII, and §22577(a)(7), which covers information types that only qualify as PII in combination with other identifiers.

Pract	***Acc*** *(n=20)*	***Prec***$_{pos}$ *(n=20)*	***Rec***$_{pos}$ *(n=20)*	***F-1***$_{pos}$ *(n=20)*	***Inconsistent*** *(n=20)*
CID	0.85	**0.5**	1	0.67	15%
CL	0.75	**0.38**	1	0.55	15%
CC	0.95	1	0.75	0.86	15%
SID	0.95	0.94	1	0.97	75%
SL	0.9	0.75	1	0.86	30%
SC	1	-	-	-	0%

TABLE VII: *Classification results for the Cal AG app/policy set (n=20).*

We evaluated our system implementation for the Cal AG on a random sample of 20 popular Play store apps and their associated policies (Cal AG app/policy set). We asked the Cal AG users to give us their interpretations of the policies and used these to evaluate the results of our system. As shown in Table VII, it appears that the Facebook Login functionality is able to identify the contact information collection practice fairly reliably. Obviously, the results are limited to a small number of 20 app/policy pairs. Further, our system achieves high recall overall. It performs well on identifying the absence of potential inconsistencies. At the same time, similar to our results in § V-A, we find a non-negligible number of false positives. Particularly, we observe a precision of 0.5 for collection of device IDs and 0.38 for locations. However, despite these low precision values, the high recall values suggest that our system is unlikely to miss many potential inconsistencies. Rather, upon closer manual inspections some of those will prove to be false alarms.

VII. Future Directions

The law of notice and choice is intended to enable enforcement of data practices in mobile apps and other online services. However, verifying whether an app actually behaves according to the law and its privacy policy is decisively hard. To alleviate this problem we propose the use of an automated analysis system based on machine learning and static analysis to identify potential privacy requirement inconsistencies. Our system advances app privacy in three main areas: it increases transparency for otherwise largely opaque data practices, offers the scalability necessary for potentially making an impact on the app eco-system as a whole, and provides a first step towards automating mobile app privacy compliance analysis.

Our results suggest the occurrence of potential privacy requirement inconsistencies on a large scale. However, the possibilities of the techniques introduced here have yet to be fully explored. For example, the privacy policy analysis can be further developed to capture nuances in policy wording—possibly by leveraging the structure of policies (e.g., by identifying definitions of PII and where those are referenced in a policy). Similarly, the accuracy of the app analysis could be enhanced by integrating data flow analysis techniques. However, for performance reasons resources should be used

sparingly. A practical system for the purpose of large-scale app analysis necessarily remains relatively lightweight.

The findings in this study raise the question of extending our approach to other areas. We believe, the principles could be used in analyzing website practices, e.g., by leveraging the work of Sen et al. [61]. First and third party cookies and other tracking mechanisms could be observed to evaluate collection and sharing of data. Implementing our approach in other mobile platforms, particularly, iOS, is likely more challenging. Notably, the difficulty of decompiling iOS apps might necessitate a dynamic app analysis approach [19], [43]. The web interface of Apple's App Store also does not seem to provide app pages with a standardized privacy policy link field. However, these challenges would not need to be resolved for the integration of a privacy requirement analysis into iOS software development tools.

We think that it is necessary to develop the proposed privacy requirement analysis in tandem with public policy and law. Regulators are currently pushing for app store standardization [10] and early enforcement of potentially non-compliant privacy practices [31]. Approaches like the one proposed here can relieve regulators' workloads through automation allowing them to focus their limited resources to move from a purely reactionary approach towards systematic oversight. As we also think that many software publishers do not intend non-compliance with privacy requirements, but rather lose track of their obligations or are unaware of them, we also see potential for implementing privacy requirement analyses in software development tools and integrating them into the app vetting process in app stores.

Acknowledgment

We would like to thank the anonymous reviewers for their comments on the draft of this study. We are also grateful for the insights provided by our Cal AG collaborators Justin Erlich, Cassidy Kim, Stacey Schesser, TiTi Nguyen, Joanne McNabb, Sarah Dalton, Jeremy AronDine, and Sundeep Pattem. We further thank our academic collaborators Aswarth Dara, Peter Story, Mads Schaarup Andersen, Amit Levy, Vaggelis Atlidakis, and Jie SB Li. This study was supported in part by the NSF under grants CNS-1330596, CNS-1330214, and SBE-1513957, as well as by DARPA and the Air Force Research Laboratory, under agreement number FA8750-15-2-0277. The US Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the Air Force Research Laboratory, the NSF, or the US Government.

References

[1] V. Afonso, A. Bianchi, Y. Fratantonio, A. Doupe, M. Polino, P. de Geus, C. Kruegel, and G. Vigna, "Going native: Using a large-scale analysis of android apps to create a practical native-code sandboxing policy," in *NDSS '16*. ISOC.

[2] Amazon, "Amazon EC2 instance types," https://aws.amazon.com/ec2/instance-types/, accessed: Dec 19, 2016.

[3] AppBrain, "Android library statistics," http://www.appbrain.com/stats/libraries/, accessed: Dec 19, 2016.

[4] 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 *PLDI '14*. ACM.

[5] Ask Solem & Contributors, "Celery - distributed task queue," http://docs.celeryproject.org/en/latest/, accessed: Dec 19, 2016.

[6] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the android permission specification," in *CCS '12*. ACM.

[7] R. Balebako, A. Marsh, J. Lin, J. Hong, and L. F. Cranor, "The privacy and security behaviors of smartphone app developers," in *USEC '14*.

[8] R. Bhoraskar, S. Han, J. Jeon, T. Azim, S. Chen, J. Jung, S. Nath, R. Wang, and D. Wetherall, "Brahmastra: Driving apps to test the security of third-party components," in *USENIX Security '14*. USENIX Assoc.

[9] T. Book and D. S. Wallach, "A case of collusion: A study of the interface between ad libraries and their apps," in *SPSM '13*. ACM.

[10] California Department of Justice, "Attorney General Kamala D. Harris secures global agreement to strengthen privacy protections for users of mobile applications," http://www.oag.ca.gov/news/press-releases/attorney-general-kamala-d-harris-secures-global-agreement-strengthen-privacy, Feb. 2012, accessed: Dec 19, 2016.

[11] ——, "Making your privacy practices public," https://oag.ca.gov/sites/all/files/agweb/pdfs/cybersecurity/making_your_privacy_practices_public.pdf, May 2014, accessed: Dec 19, 2016.

[12] ——, "Attorney General Kamala D. Harris launches new tool to help consumers report violations of California Online Privacy Protection Act (CalOPPA)," https://oag.ca.gov/news/press-releases/attorney-general-kamala-d-harris-launches-new-tool-help-consumers-report, Oct. 2016, accessed: Dec 19, 2016.

[13] P. Chundi and P. M. Subramaniam, "An Approach to Analyze Web Privacy Policy Documents," in *KDD Workshop on Data Mining for Social Good*, 2014.

[14] E. Costante, J. den Hartog, and M. Petkovic, "What websites know about you: Privacy policy analysis using information extraction," in *Data Privacy Management '13*. Springer.

[15] E. Costante, Y. Sun, M. Petković, and J. den Hartog, "A machine learning solution to assess privacy policy completeness," in *WPES '12*. ACM.

[16] L. F. Cranor, K. Idouchi, P. G. Leon, M. Sleeper, and B. Ur, "Are they actually any different? comparing thousands of financial institutions' privacy practices," in *WEIS '13*.

[17] L. F. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. M. Reagle, "The Platform for Privacy Preferences 1.0 (P3P1.0) specification," World Wide Web Consortium, Recommendation REC-P3P-20020416, April 2002.

[18] S. Demetriou, W. Merrill, W. Yang, A. Zhang, and C. Gunter, "Free for all! assessing user data exposure to advertising libraries on android," in *NDSS '16*. ISOC.

[19] Z. Deng, B. Saltaformaggio, X. Zhang, and D. Xu, "iRiS: Vetting private api abuse in iOS applications," in *CCS '15*. ACM.

[20] A. Desnos, "Androguard," http://doc.androguard.re/html/index.html, accessed: Dec 19, 2016.

[21] G. Dumpleton, "Modwsgi," https://modwsgi.readthedocs.io/en/develop/, accessed: Dec 19, 2016.

[22] 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 *OSDI '10*. USENIX Assoc.

[23] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *USENIX Security '11*. USENIX Assoc.

[24] ESRB, "ESRB ratings guide," http://www.esrb.org/ratings/ratings_guide.aspx, accessed: Dec 19, 2016.

[25] Facebook, "Permissions reference - Facebook login," https://developers.facebook.com/docs/facebook-login/permissions, accessed: Dec 19, 2016.

[26] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *CCS '11*. ACM.

[27] FTC, "Privacy online: A report to congress," https://www.ftc.gov/reports/privacy-online-report-congress, Jun. 1998, accessed: Dec 19, 2016.

[28] ——, "Mobile apps for kids: Current privacy disclosures are disappointing," http://www.ftc.gov/os/2012/02/120216mobile_apps_kids.pdf, Feb. 2012, accessed: Dec 19, 2016.

[29] ——, "Mobile apps for kids: Disclosures still not making the grade," https://www.ftc.gov/reports/mobile-apps-kids-disclosures-still-not-making-grade, Dec. 2012, accessed: Dec 19, 2016.

[30] ——, "Mobile privacy disclosures," www.ftc.gov/os/2013/02/130201mobileprivacyreport.pdf, Feb. 2013, accessed: Dec 19, 2016.

[31] ——, "FTC warns children's app maker BabyBus about potential COPPA violations," https://www.ftc.gov/news-events/press-releases/2014/12/ftc-warns-childrens-app-maker-babybus-about-potential-coppa, Dec. 2014, accessed: Dec 19, 2016.

[32] ——, "What's the deal? a federal trade commission study on mobile shopping apps," https://www.ftc.gov/reports/whats-deal-federal-trade-commission-study-mobile-shopping-apps-august-2014, Aug. 2014, accessed: Dec 19, 2016.

[33] ——, "Kids' apps disclosures revisited," https://www.ftc.gov/news-events/blogs/business-blog/2015/09/kids-apps-disclosures-revisited, Sep. 2015.

[34] K. Ghazinour, M. Majedi, and K. Barker, "A model for privacy policy visualization," in *COMPSAC '09*. IEEE.

[35] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "AndroidLeaks: Automatically detecting potential privacy leaks in android applications on a large scale," in *TRUST '12*. Springer.

[36] M. I. Gordon, D. Kim, J. Perkins, L. Gilham, N. Nguyen, and M. Rinard, "Information-flow analysis of Android applications in DroidSafe," in *NDSS '15*. ISOC.

[37] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *ICSE '14*. ACM.

[38] GPEN, "2014 annual report," https://www.privacyenforcement.net/node/513, Mar. 2015, accessed: Dec 19, 2016.

[39] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *WISEC '12*. ACM.

[40] C. Hoke, L. Cranor, P. Leon, and A. Au, "Are They Worth Reading? An In-Depth Analysis of Online Trackers Privacy Policies," *I/S : A Journal of Law and Policy for the Information Society*, Apr. 2015.

[41] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang, "AsDroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction," in *ICSE '14*. ACM.

[42] D. Kong, L. Cen, and H. Jin, "AUTOREB: Automatically understanding the review-to-behavior fidelity in android applications," in *CCS '15*. ACM.

[43] A. Kurtz, A. Weinlein, C. Settgast, and F. Freiling, "DiOS: Dynamic privacy analysis of ios applications," Friedrich-Alexander-Universität Erlangen-Nürnberg, Dept. of Computer Science, Tech. Rep. CS-2014-03, Jun. 2014.

[44] J. Lin, B. Liu, N. Sadeh, and J. I. Hong, "Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings," in *SOUPS '14*. USENIX Assoc.

[45] B. Liu, B. Liu, H. Jin, and R. Govindan, "Efficient privilege de-escalation for ad libraries in mobile apps," in *MobiSys '15*. ACM.

[46] F. Liu, R. Ramanath, N. Sadeh, and N. A. Smith, "A step towards usable privacy policy: Automatic alignment of privacy statements," in *COLING '14*.

[47] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[48] A. K. Massey, J. Eisenstein, A. I. Antón, and P. P. Swire, "Automated text mining for requirements analysis of policy documents," in *RE '13*.

[49] K. Olmstead and M. Atkinson, "Apps permissions in the Google Play store," http://www.pewinternet.org/2015/11/10/apps-permissions-in-the-google-play-store/, Nov. 2015, accessed: Dec 19, 2016.

[50] Onyxbits, "Raccoon - apk downloader," http://www.onyxbits.de/raccoon, accessed: Dec 19, 2016.

[51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, 2011.

[52] Pivotal Software, Inc, "Rabbitmq," https://www.rabbitmq.com/, accessed: Dec 19, 2016.

[53] ponty, "Pyvirtualdisplay," https://pypi.python.org/pypi/PyVirtualDisplay, accessed: Dec 19, 2016.

[54] Progress Software Corporation, "Fiddler," http://www.telerik.com/fiddler, accessed: Dec 19, 2016.

[55] R. Ramanath, F. Liu, N. Sadeh, and N. A. Smith, "Unsupervised alignment of privacy policies using hidden markov models," in *ACL '14*.

[56] J. R. Reidenberg, T. Breaux, L. F. Cranor, B. French, A. Grannis, J. T. Graves, F. Liu, A. McDonald, T. B. Norton, R. Ramanath, N. C. Russell, N. Sadeh, and F. Schaub, "Disagreeable privacy policies: Mismatches between meaning and users' understanding," *Berkeley Technology Law Journal*, vol. 30, no. 1, pp. 39–88, 2015.

[57] D. Reidsma and J. Carletta, "Reliability measurement without limits," *Comput. Linguist.*, vol. 34, no. 3, pp. 319–326, Sep. 2008.

[58] A. Ronacher, "Flask," http://flask.pocoo.org/, accessed: Dec 19, 2016.

[59] N. Sadeh, A. Acquisti, T. D. Breaux, L. F. Cranor, A. M. McDonald, J. R. Reidenberg, N. A. Smith, F. Liu, N. C. Russell, F. Schaub, and S. Wilson, "The usable privacy policy project," Carnegie Mellon University, Tech. report CMU-ISR-13-119, 2013.

[60] Selenium project, "Selenium," http://www.seleniumhq.org/, accessed: Dec 19, 2016.

[61] S. Sen, S. Guha, A. Datta, S. K. Rajamani, J. Tsai, and J. M. Wing, "Bootstrapping privacy compliance in big data systems," in *SP '14*.

[62] R. Slavin, X. Wang, M. Hosseini, W. Hester, R. Krishnan, J. Bhatia, T. Breaux, and J. Niu, "Toward a framework for detecting privacy policy violation in android application code," in *ICSE '16*. ACM.

[63] J. W. Stamey and R. A. Rossi, "Automatically identifying relations in privacy policies." in *SIGDOC '09*. ACM.

[64] The Apache Software Foundation, "Apache," http://httpd.apache.org/, accessed: Dec 19, 2016.

[65] C. Tumbleson and R. Wiśniewski, "Apktool," https://ibotpeaches.github.io/Apktool/, accessed: Dec 19, 2016.

[66] T. Watanabe, M. Akiyama, T. Sakai, and T. Mori, "Understanding the inconsistencies between text descriptions and the use of privacy-sensitive resources of mobile apps," in *SOUPS '15*. USENIX Assoc.

[67] S. Wilson, F. Schaub, A. A. Dara, F. Liu, S. Cherivirala, P. G. Leon, M. S. Andersen, S. Zimmeck, K. M. Sathyendra, N. C. Russell, T. B. Norton, E. Hovy, J. Reidenberg, and N. Sadeh, "The creation and analysis of a website privacy policy corpus," in *ACL '16*. ACL.

[68] S. Wilson, F. Schaub, R. Ramanath, N. Sadeh, F. Liu, N. A. Smith, and F. Liu, "Crowdsourcing annotations for websites' privacy policies: Can it really work?" in *WWW '16*.

[69] L. Yu, X. Luo, X. Liu, and T. Zhang, "Can we trust the privacy policies of android apps?" in *DSN '16*.

[70] L. Yu, T. Zhang, X. Luo, and L. Xue, "AutoPPG: Towards automatic generation of privacy policy for android applications," in *SPSM '15*. ACM.

[71] M. Zhang, Y. Duan, Q. Feng, and H. Yin, "Towards automatic generation of security-centric descriptions for android apps," in *CCS '15*. ACM.

[72] S. Zimmeck and S. M. Bellovin, "Privee: An architecture for automatically analyzing web privacy policies," in *USENIX Security '14*. USENIX Assoc.

APPENDIX A
POLICY AND APP DATASETS

1) Full App Set — Total Apps Collected from the Google Play Store (n=17,991)
2) Full Policy Set — Total Policies Collected for Apps in the Full App Set via Google Play Store Links (n=9,295)
3) Full App/Policy Set — Total App/Policy Pairs from the Full App and Full Policy Sets adjusted for Links not actually leading to a Policy (n=9,050)
4) Policy Test Set — Random Policies from the OPP-115 Corpus [67] (n=40)
5) App Test Set — Random Apps Associated with the Policies in the Policy Test Set (n=40)
6) App/Policy Test Set — App/Policy Pairs from the App and Policy Test Sets (n=40)
7) Cal AG App/Policy Set — Random Popular Apps and Associated Policies from the Google Play Store (n=20)

APPENDIX B
EVALUATING POTENTIAL INCONSISTENCIES

Statistic	***Policy and App Classification***
TP	Policy=0 correct and App=1 correct
FP	**Policy=0 incorrect and App=1 correct**, or Policy=0 correct and App=1 incorrect, or Policy=0 incorrect and App=1 incorrect
FN	Policy=1 incorrect and App=0 incorrect, or Policy=1 incorrect and App=1 correct, or Policy=0 correct and App=0 incorrect
TN	All remaining combinations

TABLE VIII: *Evaluating potential inconsistencies. For example, a false positive can be the result of a policy being incorrectly classified as not covering a practice (Policy=0) while the practice actually occurs in the corresponding app (App=1), which was correctly identified.*