Actions Speak Louder than Words: Entity-Sensitive Privacy Policy and Data Flow Analysis with POLICHECK

Benjamin Andow,[⋆] Samin Yaseer Mahmud,[†] Justin Whitaker,[†]
William Enck,[†] Bradley Reaves,[†] Kapil Singh,[⋆] Serge Egelman[‡]
[⋆]*IBM T.J. Watson Research Center*
[†]*North Carolina State University*
[‡]*U.C. Berkeley / ICSI / AppCensus Inc.*

Abstract

Identifying privacy-sensitive data leaks by mobile applications has been a topic of great research interest for the past decade. Technically, such data flows are not "leaks" if they are disclosed in a privacy policy. To address this limitation in automated analysis, recent work has combined program analysis of applications with analysis of privacy policies to determine the flow-to-policy consistency, and hence violations thereof. However, this prior work has a fundamental weakness: it does not differentiate the *entity* (e.g., first-party vs. third-party) receiving the privacy-sensitive data. In this paper, we propose POLICHECK, which formalizes and implements an entity-sensitive flow-to-policy consistency model. We use POLICHECK to study 13,796 applications and their privacy policies and find that up to 42.4% of applications either incorrectly disclose or omit disclosing their privacy-sensitive data flows. Our results also demonstrate the significance of considering entities: without considering entity, prior approaches would falsely classify up to 38.4% of applications as having privacy-sensitive data flows consistent with their privacy policies. These false classifications include data flows to third-parties that are omitted (e.g., the policy states only the first-party collects the data type), incorrect (e.g., the policy states the third-party does not collect the data type), and ambiguous (e.g., the policy has conflicting statements about the data type collection). By defining a novel automated, entity-sensitive flow-to-policy consistency analysis, POLICHECK provides the highest-precision method to date to determine if applications properly disclose their privacy-sensitive behaviors.

1 Introduction

Privacy is a long-standing open research challenge for mobile applications. Literature has proposed various program analysis tools for Android [7, 11, 12, 14] and iOS [10] apps, often citing private-information disclosure as motivations. Subsequent empirical studies [16, 18, 23–25, 27] have demonstrated pervasive and continual disclosure of privacy-sensitive information such as device identifiers and geographic location.

Broadly speaking, the concept of privacy is only vaguely defined and frequently debated. Privacy resides at the intersection of technical, cultural, and legal considerations. In the case of mobile applications, data collection and sharing are often considered (legally) acceptable if it is disclosed in the privacy policy for the application. While there have been several manual analyses of application privacy policies [9, 20], it is hard to computationally reason about what privacy policies say, and therefore how applications adhere to them.

A recent thread of research has begun studying privacy policies and mobile applications [29, 32, 34, 38]. The goal of these studies is to help app developers write accurate privacy policies, help application stores identify privacy violations, and help end users choose more-privacy-friendly applications. Conceptually, these studies use a combination of static program analysis and natural language processing to perform an analysis of *flow-to-policy consistency*. Simply, flow-to-policy consistency analysis determines whether an app's behavior is consistent with what is declared in the privacy policy.

While such prior studies have led to promising results, the techniques have a fundamental weakness: they do not differentiate the *entity* (e.g., first-party vs. third-parties, such as advertisers and analytics providers) receiving the data. In fact, in Section 5.2, we show that entity-insensitive models may wrongly classify 38.4% of applications as having privacy-sensitive data flows consistent with their privacy policies due to reasoning over third-party data flows using policy statements discussing first-party collection practices. For example, consider the following sentence from a popular Android application with over 10 million downloads:

> *When you launch any of our applications, we collect information regarding your device type, operating system and version, carrier provider, IP address, Media Access Control (MAC) address, International Equipment Mobile ID (IMEI), whether you are using a point package, the game version, the device's geo-location, language settings, and unique device ID.*

This statement indicates that the app (the first-party) collects different device identifiers; but there is no mention of third-parties collecting this data. In actuality, dynamic analysis found that the application sends the IMEI, Android ID, and Ad ID to Tapjoy and the Android ID and Ad ID to Flurry (two third-party advertisers). By not considering the entity receiving the privacy-sensitive data, prior work would incorrectly classify these data flows as being consistent with the policy.

In addition, the importance developers disclosing the third-party entities with which they are sharing information is grounded in regulations, such as GDPR [3] and CCPA [1]. In particular, GDPR mandates that data controllers disclose the recipients or categories of recipients with which they share personal data. In the case of applications, the first-party (developer) is considered the data controller while third-parties can either be data controllers or data processors. The majority of the entities involved in this study self-identify as data controllers (e.g., Google, Facebook, TapJoy). Based on the requirement that data controllers are required to disclose their identity and contact information according to GDPR, it is debatable whether the application is required to disclose all third-parties by name if they also take the role as a data controller. Further, the CCPA states that the privacy policy should disclose the categories of third-parties with whom the business shared personal information. Therefore, the application's privacy policy is also mandated to disclose the third-party entities with which they share data based on the CCPA.

In this paper, we propose POLICHECK, which provides an *entity-sensitive* flow-to-policy consistency model to determine if an application's privacy policy discloses relevant data flows. We formally specify a novel flow-to-policy consistency model that is sensitive to the semantic granularity of both the data type and the entity receiving the data and sentiment of the statement. We dissect flow-to-policy consistency into 5 distinct types of disclosures (including non-disclosures) to allow for targeted exploration of how apps are (not) disclosing their privacy practices. We use POLICHECK to study the flow-to-policy consistency of 13,796 Android applications observed to send privacy sensitive values to servers during dynamic analysis (45,603 data flows).

The findings from our large-scale empirical study found several significant flow-to-policy inconsistencies in popular real-world applications that impact tens-of-millions of users, such as not disclosing data sharing with advertisers and analytics providers in privacy policies. In general, we found that applications almost never clearly disclose their privacy-sensitive data flows to third-parties. In fact, 40.4% of data flows involving third-party entities are broadly discussed using the term "third-party," leaving it up to guesswork to determine where the data is flowing. Furthermore, we found 5.2% of applications state that they do not share or collect a specific type of information within their privacy policy, but dynamic analysis shows the opposite.

The results from our empirical study highlight the poor state of privacy policies for Android applications, which demonstrates the need for action from regulatory agencies and application markets. For example, the FTC has set precedent by charging mobile applications that were found to be omitting or incorrectly disclosing their privacy practices [15, 30], which corresponds to our omitted disclosures and incorrect disclosures. Regulatory agencies could use POLICHECK for automated analysis at-scale to identify applications violating their privacy policies and take whichever actions they deem appropriate. Further, application markets could also leverage POLICHECK to triage and remove applications that are not correctly disclosing their privacy practices and to urge developers to provide clearer disclosures.

This paper makes the following main contributions:

- *We formally define an* entity-sensitive *flow-to-policy consistency model for mobile apps.* This model includes two types of consistencies and three types of inconsistencies. By considering entities, the model avoids significant *misclassifications* that result from prior approaches.

- *We design and implement the* POLICHECK *tool for analyzing the flow-to-policy consistency of Android applications.* POLICHECK builds on top of PolicyLint [4] for privacy policy analysis and AppCensus [6] for dynamic analysis of Android applications. In doing so, we bridge the gap between the low-level data types and DNS domains used by program analysis tools and the often higher-level concepts present in privacy policies.

- *We study and characterize the flow-to-policy consistency of 13,796 Android applications*. Our characterization differentiates first-party and third-party collection and demonstrates the importance of an entity-sensitive consistency model. We show that our entity-sensitive consistency finds significant flow-to-policy inconsistencies that involve sharing data to third-party entities, impacting tens-of-millions of users.

The rest of this paper proceeds as follows. Section 2 uses examples to provide the high-level intuition behind POLICHECK. Section 3 formally defines the different types of flow-to-policy consistencies and inconsistencies. Section 4 describes the design of POLICHECK. Section 5 presents our empirical study. Section 6 discusses additional case studies. Section 7 discusses limitations and future work. Section 8 overviews related work. Section 9 concludes.

2 Flow-to-Policy Consistency

This section motivates POLICHECK's functionality through five examples that POLICHECK identified. We simultaneously provide a high-level intuition of its functionality by walking through how a human analyst might approach the task. In doing so, we also exemplify the limitations of prior work.

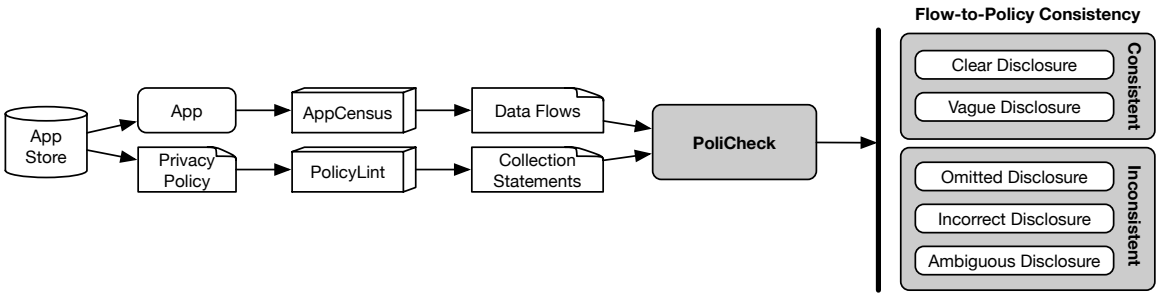

Figure 1: POLICHECK determines the consistency of a mobile application's data flows to its privacy policy.

This section does not cover every corner case. Sections 3 and 4 describe POLICHECK in detail.

As shown in Figure 1, POLICHECK seeks to determine if the privacy policies for mobile applications disclose their privacy-sensitive data flows to different network entities, as required by various regulations [1, 3]. We define a data flow as a type of privacy-sensitive data (e.g., IMEI, location, email address) and the entity receiving the data (e.g., Facebook, TapJoy, AdMob). If an application's privacy policy appropriately discusses the sharing or collection of the specific data type to or by a specific entity for a given data flow, we refer to the data flow as being *consistent* with the privacy policy. To ensure sufficient evidence of sharing or collection by an entity, we scope data flows to network transmission identified during dynamic analysis. While dynamic analysis may under-approximate data flows if sufficient code coverage is not achieved during testing, our goal was to optimize for precision over recall. In contrast, static analysis may over-approximate data flows and lead to lower precision (e.g., some ad libraries collect geographic location based on an application developer's server-side configuration).

2.1 Clear Disclosures

A data flow has a *clear disclosure* when there exists a statement within the privacy policy that directly discusses the exact type of data and entity of the data flow, *and* there is no other policy statement that contradicts it. For illustrative purposes, consider the "Dr. Panda Town: Vacation" (`com.drpanda.town.holiday`) game application with over 1 million downloads on Google Play. This application is built on top of the Unity third-party game engine. For analytics purposes, it obtains the device's advertising identifier and sends it to `cdp.cloud.unity3d.com` (i.e., Unity).

To determine if this data flow is disclosed by the privacy policy, the first step is to resolve `cdp.cloud.unity3d.com` to the entity "Unity" by matching the root domain (`unity3d.com`) to a list of known analytics providers. For each policy statement, we look for a direct positive sentiment match between the flow's data type and entity and the policy statement's data type and entity. In this case, we identify the following statement, "*Unity collects the following information through our Games: unique device ID and AD ID.*" We then look for policy statements that contradict the statement by extracting all negative sentiment statements that discuss the flow's data type and entity at any semantic granularity (e.g., analytics providers collecting device information). In this specific case, we do not find any policy statements that contradict the statement above. Therefore, we label this case as a clear disclosure.

2.2 Vague Disclosures

A data flow has a *vague disclosure* when the only statements within a privacy policy that match a data flow use broad terms for the data type or entity. Similar to clear disclosures, a statement is a vague disclosure only if a contradictory policy statement does not exist. We differentiate vague disclosures from clear disclosures, because there is a risk that the language used to disclose the data flow is so broad that it encapsulates a wide-range of data flows, making it difficult to determine if third-party sharing or collection occurs. Vague disclosures are similar to Slavin et al. [29] and Wang et al.'s [32] definition of weak violations, but entity-sensitive, sentiment-sensitive, and contradiction-sensitive.

As an example, consider the popular "Elite Killer: SWAT" (`com.yx.sniper`) game application on Google Play with over 10 million downloads and a 4.3 star rating. For monetization purposes, this application uses the TapJoy advertising provider to deliver advertisements within the application. When requesting advertisements from TapJoy, the application obtains the user's Android advertising identifier and transmits it to `ws.tapjoyads.com`.

Similar to the previous example, we resolve `ws.tapjoyads.com` to "TapJoy" through a substring match of the root domain in our list of known advertisers. However, rather than identifying only direct matches, we look for policy statements with positive sentiment that match at any semantic granularity for the flow's data type and entity. In this case, we identify the following statement, "*A device identifier and in-game or user session activity may be shared with the advertiser.*" This statement matches the data flow, because TapJoy is an advertiser and the Android advertising identifier is a type of device identifier. Next, we look for matching policy statements with negative sentiment statements. Since we do not

find any policy statements that contradict this statement, we label this data flow as a vague disclosure. Finally, we calculate a vagueness score for the resolved policy statement to allow a ranked ordering, which is based on a normalized ontological distance between the flow's data type and entity and the policy statements data type and entity.

2.3 Omitted Disclosure

A data flow has an *omitted disclosure* when there are no policy statements that discuss it. Omitted disclosures are similar to Wang et al.'s [32] definition of strong violations. However, as we demonstrate in the following example, prior definitions do not consider both data type and entity, and therefore may incorrectly classify an omitted disclosure as being flow-to-policy consistent.

Consider the application "Flash Emoji Keyboard & Themes" (`com.xime.latin.lite`) on Google Play, which currently has over 50 million downloads and a 4.1 star rating. This application uses the Avazu advertising provider to serve advertisements within the application for monetization purposes. When requesting advertisements from Avazu, this application obtains the user's Android identifier, IMEI, and phone number and transmits that information to Avazu servers (`api.c.avazunativeads.com`).

Similar to the previous cases, we look for policy statements that describe the data flow at any semantic granularity, but with both positive and negative sentiment. For these data flows, we do not find any policy statements that match both data type and entity.

This example application demonstrates the need for considering both the data type and entity. If we only considered the data type, we would identify the following policy statement: "*When you access our Services, we automatically record and upload information from your device including, but not limited to attributes such as the operating system, hardware version, device settings, battery and signal strength, device identifiers...*" This policy statement indicates application itself is collecting device identifiers. However, it does not disclose a data flow to the advertiser. Therefore, the Android identifier, IMEI, and phone number data flows to the advertiser lack flow-to-policy consistency. Prior works [29,32,38] that do not consider entities when reasoning over privacy policies would have incorrectly identified these data flows as consistent.

2.4 Incorrect Disclosure

A data flow has an *incorrect disclosure* if a policy statement indicates that the flow will *not* occur (i.e., a negative sentiment sharing or collection statement) and there is not a contradicting positive sentiment statement. However, we must be careful when determining if a positive sentiment policy statement contradicts the negative sentiment statement. PolicyLint [4] identified a class of *narrowing definitions* (labeled N_1 to N_4 in Table 1) that use a negative sentiment when referring to a *more specific* data type or entity. A human reading these policy statements would not view them as contradicting; rather, the negative sentiment statement would be viewed as providing an exception to a broad sharing or collection practice. Therefore, we still classify a data flow as an incorrect disclosure if there is a corresponding positive sentiment statement that matches the narrowing definitions relationship.

For example, consider the "Furby BOOM!" (`com.hasbro.FurbyBoom`) game application, which has over 10 million downloads on Google Play. This application is built on top of the Unity third-party game engine. To provide statistics to Unity for optimization purposes of their game platform, the application obtains and sends the device's IMEI to Unity (`stats.unity3d.com`).

Similar to the above cases, we find all relevant policy statements that describe the data flow at any semantic granularity. In this case, we only find one policy statement, "*Our Apps do not send the device ID or IP address to us or to any third-party, and our App does not make further use of this information.*" As the device's IMEI is a type of device identifier and Unity is a third-party, the application is inconsistent with its own policy, as it is stating that the data flow should not exist. Note that prior works [29,32] would have incorrectly identified this data flow as consistent with the policy, as they do not handle negative sentiment statements.

2.5 Ambiguous Disclosure

A data flow has an *ambiguous disclosure* if the flow matches two or more contradictory policy statements where it is not clear if the flow will or will not occur. As mentioned above, PolicyLint [4] identified different types of relationships between positive and negative sharing statements. We classify a data flow as having an ambiguous disclosure if there exist two policy statements that have a *logical contradiction* relationship (C_1 to C_5 in Table 1), but not a narrowing definition relationship (N_1 to N_4 in Table 1), as described above. Furthermore, we introduce a new set of conflicting policy statements called *flow-sensitive contradiction* relationships (C_6 to C_{12} in Table 1), which Section 3 explains in more detail. Data flows matching two or more policy statements with a flow-sensitive contradiction relationship are also classified as ambiguous disclosures.

For example, consider the "Flip Diving" (`com.motionvolt.flipdiving`) game application, which has over 50 million downloads on Google Play. This application uses the AdColony advertising provider to serve advertisements for monetization purposes. When requesting advertisements from AdColony, the application obtains the user's Android advertising identifier and transmits it to `androidads23.adcolony.com`.

Similar to the above cases, we find all relevant policy statements that describe the data flow at any semantic granularity. In this case, we only find two relevant policy statements, "*On*

our apps, these third party advertising companies will collect and use your data to provide you with targeted advertising that is relevant to you and your preferences with your consent." and "*We don't give or sell your data to third parties for them to market to you.*" As their policy states that they both do and do not give your data to third-parties for advertising/marketing, the policy is ambiguous. Prior works [29, 32] would have falsely identified this data flow as consistent, as they do not capture negative sentiment sharing or collection statements, nor do they identify conflicting statements.

3 Consistency Model

In this section, we provide the core logic model for our definition of flow-to-policy consistency, as motivated in Section 2. We begin with a formal specification of data flows and privacy policy statements. We then introduce four ontological operations required for reasoning over data flows. Finally, we formalize the two types of flow-to-policy consistencies (clear disclosures and vague disclosures) and the three types of inconsistencies (omitted disclosures, incorrect disclosures, and ambiguous disclosures).

3.1 Data Flow and Policy Statements

We model an application a as a tuple, $a = (F, P)$, where F is a set of data flows observed for the application and P is a set of sharing and collection policy statements extracted from the application's privacy policy. Let D represent the total set of data types and E represent the total set of entities. Then, a data flow is represented by the following definition.

Definition 3.1 (Data Flow)**.** A data flow $f \in F$ is a tuple $f = (e, d)$ where $d \in D$ is the data type that is sent to an entity $e \in E$.

For example, an application that sends the device's advertising identifier to AdMob can be concisely represented by the data flow tuple (AdMob, advertising identifier).

Similar to PolicyLint [4], we represent a sharing and collection policy statement as a tuple (*actor*, *action*, *data type*, *entity*) where the *actor* performs an *action* on a *data type*, and an *entity* receives a data object of that type. We consider four actions: *share*, *not share*, *collect*, and *not collect*. For example, the statement, "*We will share your personal information with advertisers*" is represented as (we, share, personal information, advertisers). As our analysis can only observe client-side behaviors, we adopt PolicyLint's simplified policy statement form, which transforms the 4-tuple into a more compact 3-tuple. Intuitively, these transformation rules remove the actor and sharing actions and only considers the entities who may possibly receive (i.e., collect) the data type based on the policy statement (e.g., sharing data implies the actor also collects it). Therefore, we represent a policy statement as follows.

Definition 3.2 (Policy Statement)**.** A policy statement $p \in P$ is a tuple, $p = (e, c, d)$, where data type $d \in D$ is either collected or not collected, $c \in \{\text{collect}, \text{not\_collect}\}$, by an entity $e \in E$.

For example, the above 4-tuple (we, share, personal information, advertisers) is represented as two 3-tuples: (we, collect, personal information) and (advertisers, collect, personal information).

3.2 Ontological Operations

Privacy policies may disclose data flows using terms with a different semantic granularity than the actual data flow. For example, a privacy policy may specify (advertiser, collect, device identifier) to disclose the data flow (AdMob, advertising identifier). To match the policy statement to the data flow, an analysis tool must know that AdMob is an advertiser and an advertising identifier is a type of device identifier. These relationships are commonly referred to as subsumptive relationships, where a more specific term is subsumed under a more general term (e.g., AdMob is subsumed under advertisers, and advertising identifier is subsumed under device identifier). Such relationships are often encoded into an *ontology*, which is a rooted directed acyclic graph where terms are nodes and edges are labeled with the relationship between those terms.

Our analysis uses two ontologies: data type and entity. While ontologies can represent several different types of relationships, our ontologies are limited to subsumptive and synonym relationships. We use the following notation to describe binary relationships between terms in a given ontology, which expands on the operators defined by PolicyLint. The operators are parameterized with an ontology o, which represents either the data type (δ) or entity (ε) ontologies.

Definition 3.3 (Semantic Equivalence)**.** Let x and y be terms partially ordered by an ontology o. $x \equiv_o y$ is true if x and y are synonyms, defined with respect to an ontology o.

Definition 3.4 (Subsumptive Relationship)**.** Let x and y be terms partially ordered by "is-a" relationships in an ontology o. $x \sqsubset_o y$ is true if term x is subsumed under the term y and $x \not\equiv_o y$ (e.g., "x is-a y" or "x is-a ... is-a y"). Similarly, $x \sqsubseteq_o y \Leftrightarrow x \sqsubset_o y \ \vee \ x \equiv_o y$.

In addition to these two ontological operators, we identify a third type of ontological operator that impacts flow-to-policy consistency analysis. We define a *semantic approximation* operator that identifies terms that have common descendants in the ontology, but are not direct descendants of one other. For example, consider we have the data flow (Flurry, advertising identifier) and the policy statements: (advertiser, not_collect, identifiers) and (analytic provider, collect, identifier). As Flurry is both an advertiser and analytics provider (common descendant), the policy becomes ambiguous when

considering whether the data flow is disclosed by the policy. We define semantic approximation as follows.

Definition 3.5 (Semantic Approximation)**.** Let x and y be terms partially ordered by "is-a" relationships in an ontology o. $x \approx_o y$ is true if $\exists z$ such that $z \sqsubset_o x \wedge z \sqsubset_o y \wedge x \not\sqsubseteq_o y \wedge y \not\sqsubseteq_o x$.

Finally, when discussing vague disclosures, it is useful to characterize the vagueness using a metric. To help define this metric, we define the following two operations to determine a distance between two terms in a given ontology.

Definition 3.6 (Ontological Distance)**.** Let x and y be terms partially ordered by "is-a" relationships in an ontology o, and $x \sqsubseteq_o y$. The ontological distance $\Delta_o(x,y)$ is the shortest path between x and y.

Definition 3.7 (Normalized Ontological Distance)**.** Let x and y be terms partially ordered by "is-a" relationships in an ontology o, and $x \sqsubseteq_o y$. The normalized ontological distance $\hat{\Delta}_o(x,y)$ is the length of the shortest path between x and y divided by the length of the shortest path between x and the root node ($\top$) that goes through y. More specifically, $\hat{\Delta}_o(x,y) = \frac{\Delta_o(x,y)}{\Delta_o(x,y)+\Delta_o(y,\top)}$.

3.3 Consistency

Section 2 informally defined five types of disclosures used to describe flow-to-policy consistency and inconsistency. This section formally defines a consistency model via the logical relationships between terms in data flows and policy statements. A key part of the informal definitions in Section 2 is the interpretation of situations with conflicting policy statements, that is, contradictions and narrowing definitions. The existence of such policy statement conflicts requires flow-to-policy consistency analysis to consider the policy as a whole, rather than looking for the existence of any sharing or collection statement, as done in prior work [29, 32, 38].

PolicyLint [4] introduced five types of *logical contradictions* (C_1 to C_5) and four types of *narrowing definitions* (N_1 to N_4) as shown in Table 1. Logical contradictions are a pair of policy statements that are either exact contradictions (C_1) or those that discuss not collecting broad types of data, but also discuss collecting exact or more specific types (C_2 to C_5). Narrowing definitions are a pair of policy statements where broad data types are stated to be collected, and specific data types are stated to not be collected (N_1 to N_4).

We introduce a third pairing of conflicting policy statements called *flow-sensitive contradictions* (C_6 to C_{12} in Table 1). Flow-sensitive contradictions are a pair of policy statements with opposing sentiment, such that at least one of the data types or entities are semantically approximate to the other. Similar to logical contradictions, flow-sensitive contradictions result in an ambiguous policy when reasoning whether a specific data flow is disclosed. For example, consider we have the data flow (Flurry, advertising identifier) and the policy statements (analytic provider, collect, advertising identifier) and (advertiser, not_collect, advertising identifier). Since Flurry is both an advertiser and analytic provider, the policy is ambiguous with respect to this data flow.

Table 1: Types of conflicting policy statements in a privacy policy: narrowing definitions (N_{1-4}) and logical contradictions from PolicyLint [4], and flow-sensitive contradictions (C_{6-12}). C_{1-12} may lead to ambiguous policies.

Rule	Logic	Example*
N_1	$e_i \equiv_\varepsilon e_j \wedge d_k \sqsupset_\delta d_l$	(Flurry, collect, Dev Info) (Flurry, not_collect, IMEI)
N_2	$e_i \sqsubset_\varepsilon e_j \wedge d_k \sqsupset_\delta d_l$	(Flurry, collect, Dev Info) (Advertiser, not_collect, IMEI)
N_3	$e_i \sqsupset_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(Advertiser, collect, IMEI) (Flurry, not_collect, IMEI)
N_4	$e_i \sqsupset_\varepsilon e_j \wedge d_k \sqsupset_\delta d_l$	(Advertiser, collect, Dev Info) (Flurry, not_collect, IMEI)
C_1	$e_i \equiv_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(Flurry, collect, IMEI) (Flurry, not_collect, IMEI)
C_2	$e_i \equiv_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(Flurry, collect, IMEI) (Flurry, not_collect, Dev Info)
C_3	$e_i \sqsubset_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(Flurry, collect, IMEI) (Advertiser, not_collect, IMEI)
C_4	$e_i \sqsubset_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(Flurry, collect, IMEI) (Advertiser, not_collect, Dev Info)
C_5	$e_i \sqsupset_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(Advertiser, collect, IMEI) (Flurry, not_collect, Dev Info)
C_6	$e_i \equiv_\varepsilon e_j \wedge d_k \approx_\delta d_l$	(Flurry, collect, Dev Info) (Flurry, not_collect, Track Info)
C_7	$e_i \sqsubset_\varepsilon e_j \wedge d_k \approx_\delta d_l$	(Flurry, collect, Dev Info) (Advertiser, not_collect, Track Info)
C_8	$e_i \sqsupset_\varepsilon e_j \wedge d_k \approx_\delta d_l$	(Advertiser, collect, Dev Info) (Flurry, not_collect, Track Info)
C_9	$e_i \approx_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(Analytic, collect, IMEI) (Advertiser, not_collect, IMEI)
C_{10}	$e_i \approx_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(Analytic, collect, IMEI) (Advertiser, not_collect, Dev Info)
C_{11}	$e_i \approx_\varepsilon e_j \wedge d_k \sqsupset_\delta d_l$	(Analytic, collect, Dev Info) (Advertiser, not_collect, IMEI)
C_{12}	$e_i \approx_\varepsilon e_j \wedge d_k \approx_\delta d_l$	(Analytic, collect, Dev Info) (Advertiser, not_collect, Track Info)

*$P = \{(e_i, \text{collect}, d_k), (e_j, \text{not\_collect}, d_l)\}$, $f = (Flurry, IMEI)$

Before defining our flow-to-policy consistency model, we define three filters on the set policy statements in a policy. These filters simplify the notation used to formally describe the five disclosure types. The following discussion assumes the analysis of an individual application, and each disclosure is described with respect to a specific data flow f. Applications may have multiple data flows. Furthermore, each appli-

cation has a set of policy statements P.

Definition 3.8 (Contradicting Policy Statements)**.** Let P be a set of policy statements (Definition 3.2). P_C is the set of policy statements $p \in P$ for which there exists a $p' \in P$ such that p and p' have a logical contradiction (C_{1-5}) or a flow-sensitive contradiction (C_{6-12}).

Definition 3.9 (Narrowing Definition Policy Statements)**.** Let P be a set of policy statements (Definition 3.2). P_N is the set of policy statements $p \in P$ for which there exists a $p' \in P$ such that p and p' have a narrowing definition (N_{1-4}).

Definition 3.10 (Flow-Relevant Policy Statements)**.** Let P be a set of policy statements (Definition 3.2) and f be a data flow (Definition 3.1). P_f is the set of policy statements in P that are relevant to the data flow f. More specifically, $P_f = \{p \mid p \in P \wedge f.d \sqsubseteq_\delta p.d \wedge f.e \sqsubseteq_\varepsilon p.e\}$.

3.3.1 Flow-to-Policy Consistency

Using the above definitions, we can now define flow-to-policy consistency. As discussed in Section 2, there are two types of consistent disclosures: clear disclosures and vague disclosures. We now formally define flow-to-policy consistency and the two types of disclosures, as well as a vagueness metric to help quantify the significance of vague disclosures.

Definition 3.11 (Flow-to-Policy Consistency)**.** A data flow f is consistent with an application's privacy policy P if and only if $\exists p \in P_f$ such that $p.c = \text{collect} \wedge \nexists p' \in P_f$ such that $p'.c = \text{not\_collect}$.

Definition 3.12 (Clear Disclosure)**.** An application's privacy policy has a *clear disclosure* of a data flow f if there exists a collect policy that uses terms of the same semantic granularity for both data type and entity, *and* there does not exist a conflicting not_collect policy for the data type and entity. More specifically, there is a clear disclosure of f if and only if $\exists p \in P_f$ such that $p.c = \text{collect} \wedge f.d \equiv_\delta p.d \wedge f.e \equiv_\varepsilon p.e$ and $\nexists p' \in P_f$ such that $p'.c = \text{not\_collect}$.

Definition 3.13 (Vague Disclosure)**.** An application's privacy policy has a *vague disclosure* of a data flow f if there does not exist clear disclosure, but there does exist a collect policy using a broader semantic granularity for either the data type of entity, and there does not exist a conflicting not_collect policy for the data type and entity. More specifically, there is a vague disclosure of f if and only if $\nexists p \in P_f$ such that $p.c = \text{collect} \wedge f.d \equiv_\delta p.d \wedge f.e \equiv_\varepsilon p.e$ and $\exists p' \in P_f$ such that $p'.c = \text{collect} \wedge f.d \sqsubseteq_\delta p'.d \wedge f.e \sqsubseteq_\varepsilon p'.e$ and $\nexists p'' \in P_f$ such that $p''.c = \text{not\_collect}$.

A data flow with a vague disclosure is not necessarily bad. However, if the terms in the matching policy statement are too broad, the disclosure may not be meaningful to the user. For example, the policy statement (third-party, collect, personal data) is considerably more vague than (AdMob, collect, advertising identifier) to describe the data flow (AdMob, advertising identifier). As vagueness is subjective, we do not seek a binary classification (e.g., weak violations [29]). Instead, we provide a quantitative metric [0.0-1.0] to rank and compare statements in terms of vagueness. A higher value indicates greater vagueness.

Our metric calculates a tuple for vagueness of a flow f's disclosure via a policy statement p using the ontological distances, with values for both data type and entity. Since the magnitude of ontological distances can vary, we normalize the ontological distance to allow ranked comparisons.

Definition 3.14 (Vagueness Metric)**.** The vagueness of a flow f by a policy statement p is represented by $(\hat{\Delta}_\varepsilon(f.e, p.e), \hat{\Delta}_\delta(f.d, p.d))$.

Note that the vagueness metric allows reasoning in two-dimensions (i.e., entity and data type). We observed that reasoning in two-dimensional space increased utility of the metric for triage in comparison to reducing the metric to one-dimension by averaging of summing the scores. For example, if averaging or summing, the tuples (anyone, collect, device information) and (advertising network, collect, information) would have the same vagueness score for the flow (AdMob, Ad ID). In this case, consider an analyst wants to identify which applications are not discussing entities overly broad when disclosing data sharing practices. A one-dimensional reduction would require additional filtering based on the result, but the two-dimensional vagueness metric directly supports such triage approaches.

3.3.2 Flow-to-Policy Inconsistency

A data flow is inconsistent with the privacy policy if it does not satisfy the above consistency conditions. We define three types of disclosures that represent flow-to-policy inconsistency: omitted disclosures, incorrect disclosures, and ambiguous disclosures.

Definition 3.15 (Omitted Disclosures)**.** An application's privacy policy has an *omitted disclosure* of a data flow f if it does not include either collect or not_collect statements at any semantic granularity for the flow's data type and entity. More specifically, there is an omitted disclosure of f if and only if $P_f = \emptyset$.

Definition 3.16 (Incorrect Disclosure)**.** An application's privacy policy has an *incorrect disclosure* of a data flow f when the policy states that it does not collect or share the data type. More specifically, there is an incorrect disclosure of f if and only if $\forall p \in P_f, p.c = \text{not\_collect}$ or $(P_f \neq \emptyset \wedge P_f \cap P_N \neq \emptyset \wedge P_f \cap P_C = \emptyset)$.

Note that incorrect disclosures include narrowing definitions, because they represent an unambiguous case where a

policy has relevant flows with both collect and not_collect sentiment. Since narrowing definitions have not_collect sentiment for the more specific type, a matching data flow represents an incorrect disclosure.

Definition 3.17 (Ambiguous Disclosure)**.** An application's privacy policy has an *ambiguous disclosure* of a data flow f when it contains contradicting statements about the data flow. More specifically, there is an ambiguous disclosure of f if and only if $P_f \cap P_C \neq \emptyset$.

4 Design

The core contribution of this paper is our formalization and enhancement of flow-to-policy consistency analysis with the knowledge of which entities collect information. Determining the type of disclosure (Section 3) for each observed flow requires both dynamic analysis of applications and natural language processing of application privacy policies. We chose dynamic analysis over static analysis, because it provides (1) evidence that the flow occurs (some ad libraries use server-side configuration to determine what data types to collect), and (2) the network destination of the flow. As dynamic analysis of Android apps has received significant treatment in literature, we build upon AppCensus [6], which is the latest state-of-the-art for dynamically performing privacy analysis. Similarly, for processing privacy policies, we build on top of PolicyLint [4], which enhances prior approaches by extracting entities, as well as negative sentiment statements. PoliCheck's implementation primarily consists of our formalization provided in Section 3. The remainder of this section describes the components required to transform data flows and policy statements into our logical representation.

Data Flow Extraction: AppCensus [6] identifies privacy sensitive data flows in Android apps using the approach proposed by Reyes et al. [27]. In particular, Reyes et al. instrument the Android operating system to log access to sensitive resources and use the Android VPN API to intercept and log network traffic (including installing a root certificate to decrypt TLS traffic). They exercise the application with Monkey [5] and collect both the system and network logs. Next, they identify the privacy-sensitive data values from the system logs in the network traffic logs by using value-matching along with a set of heuristics to detect encodings of the data, such as base64 or hashing algorithms. The data flows reported by AppCensus are a tuple, (destination domain/IP address, data type). For example, the data flow discussed in Section 2.1 is represented as (`cdp.cloud.unity3d.com`, advertising identifier). Table 2 shows the complete list of data types tracked via dynamic analysis within this study.

Domain-to-Entity Mapping: While data flows are represented as a type of data being transmitted to a domain or IP address, privacy policies discuss data flows using terms for entities instead of domains (e.g., `cdp.cloud.unity3d.com` could be referred to as "Unity" within the privacy policy). Therefore, PoliCheck must map domain names to entities so that data flows conform to Definition 3.1. Note that for the data flows that had only IP addresses without domain names, we first perform a reverse-DNS search to try to resolve the IP address as a domain name. If we could not resolve a domain name, we discard the data flow from the data set.

Table 2: Data types tracked via dynamic analysis

Data Types
name, location, phone number, email address, IMEI, Wi-Fi MAC address, Ad ID, GSF ID, Android ID, serial number, SIM serial number

We curated a list of 144 advertisers and 40 analytics providers on the Google Play store from `AppBrain.com`. This list included the primary website for each organization. We manually produced a supplementary set of terms based on organization names to search our set of domain names with. After obtaining a list of potential domain names for each organization by keyword matching our search terms, we manually culled incorrect or irrelevant domain names. This resulted in a set of domain names for the top analytics and advertising organizations on Google Play.

Entity First-Party Classification: Determining which network domain is the first-party of a given application requires careful consideration. First, we check if reversing the second-level domain name of the network destination domain matches the beginning of the application's package name. For example, if the flow (advertising identifier, `analytics.mobile.walmart.com`) occurs in the app `com.walmart.android`, we mark the flow as a first-party flow, as reversing `walmart.com` results in `com.walmart`, which matches the beginning of the package name. Similarly, we check if the second-level domain name of the link to the application's privacy policy matches the root domain of the destination domain. For example, the privacy policy of the Walmart application is located at `https://corporate.walmart.com/privacy-security/walmart-privacy-policy` and the destination domain is `analytics.mobile.walmart.com`. Since the second-level domain name of the privacy policy (`walmart.com`) matches the second-level domain name domain of the destination domain is `walmart.com`, we mark the flow as first-party.

Sharing and Collection Statement Extraction: PoliCheck uses the policy statements output by PolicyLint [4]. PolicyLint uses sentence-level natural language processing to extract sharing and collection statements from privacy policies while capturing the entities and data types involved along with the sentiment of the statement. PolicyLint outputs policy statements as defined by the form in Definition 3.2.

Data Type and Entity Ontology Extension: We extended PolicyLint's ontologies to include all of the data types involved in data flows and all of the entities identified when constructing the domain-to-entity mapping. We begin by prun-

Table 3: Data Flows and Apps for each Disclosure Type

		Clear	Vague	Omit.	Incorr.	Ambig.
First	*Flows*	215	2,196	197	16	390
	Apps	205	1,589	146	9	244
Third	*Flows*	6	24,434	12,395	2,209	3,573
	Apps	6	7,105	4,582	779	990
Total	*Flows*	221	26,630	12,592	2,225	3,963
	Apps	211	7,885	4,659	788	1,193

ing PolicyLint's ontologies to remove nodes and edges that did not reach a data type or entity node in the data flows. For all missing data types and entities, we manually added them to their corresponding ontology and added edges. Finally, we extended PolicyLint's synonym list by searching policy statements using keywords for entity names and data types. For example, extended the synonym list for "advertising network" by searching the policy statements for "advertising".

Consistency Analysis: PoliCheck uses the data flows and policy statements from the prior steps to perform flow-to-policy consistency analysis. To do so, we implemented the consistency model logic defined in Section 3.

5 Consistency Characterization

Our primary motivation for creating PoliCheck was to analyze whether applications are disclosing their privacy-sensitive data flows in their privacy policies, especially for third-party sharing. In this section, we use PoliCheck to perform a large-scale study of analyzing the consistency of 45,603 data flows from 13,796 unique Android applications and their corresponding privacy policies.

Dataset Selection: To select our dataset, we began by scraping the top 100 free applications ("topselling_free" collection) across Google Play's 35 application categories in February 2019. We enhanced the dataset with an additional 42,129 randomly selected Android applications from AppCensus. Any overlaps between the two datasets were resolved by only analyzing the latest version. For each application, we downloaded the data flows from AppCensus and downloaded the HTML privacy policies from the developer's website via the link on Google Play. We excluded applications that did not have any data flows reported by AppCensus (23,488 apps). We also excluded applications whose privacy policies were not successfully downloaded (e.g., 404 errors, links to homepages) or were not written in English based on Python's `langdetect module` (6,039 apps). We also excluded data flows that did not map to nodes in our entity ontology, which resulted in a final data set of 13,796 applications with 45,603 data flows.

5.1 Consistency Analysis

We extracted sharing and collection statements from 94.4% (13,021/ 13,796) of privacy policies. From those policies, 218,257 policy statements were extracted from 48,831 sentences that were identified as a sharing or collection sentence, such that 7,526 had negative sentiment and 210,731 had positive sentiment. 31.2% (4,299/ 13,796) of policies had at least one negative sentiment policy statement and 92.6% (12,779/ 13,796) of policies had at least one positive sentiment policy statement. The policy statements discussed 34 distinct granularities of data types and 52 distinct granularities of entities. In total, there were 412 unique policy statement tuples.

For the 45,603 data flows across the 13,796 applications, there were 13 unique data types transmitted to 2,243 unique domains. The 2,243 domains were resolved to 112 unique entities. In total, there were 364 unique data flow tuples. Overall, Ad IDs were the most frequently transmitted data type, which accounted for 26,628 data flows to 100 unique entities across 11,585 applications. Across all of the flows, Unity was the most common recipient, which accounted for 6,270 data flows containing 6 unique data types across 4,381 applications.

We ran PoliCheck on the dataset and the raw statistics from analysis are listed in Table 3. We find that 42.4% of applications contain at least one omitted disclosure or incorrect disclosure, which the data flow is not disclosed by the policy or is in direct conflict with statements in the policy. We presented various case studies found by PoliCheck in Section 2. Section 6 provides additional case studies. The remainder of this section discusses the findings made possible by PoliCheck.

Note that, in this study, we consider all device IDs to be personally identifiable information (PII). Device IDs are classified as PII under GDPR [3] and CCPA [1], as they are generally used for tracking and attribution. Ad IDs were initially introduced as a pseudonymous identifier to be used to track users instead of collecting persistent identifiers, such as Android IDs, IMEI, and email addresses. However, we find that 74.7% of entities that receive Ad IDs also receive a persistent identifier. Therefore, we also consider Ad IDs to be PII, because mixing them with persistent identifiers nullifies their originally intended properties since they lose the property of non-persistence and tracking can be bridged across resets. We find that 11,589 applications send this unique identifier to third-parties, which is then linkable to users' email addresses, other device identifiers, and other sensitive information.

Finding 1: *Only 0.5% of data flows were explicitly discussed by sentences within the privacy policy in terms of the exact entity and exact data type*. In total, only 223 data flows were classified as clear disclosures. Figure 2 shows the number of clear disclosures for each of the data flow tuples. The hatched sections denote that there were no transmissions of that specific data type to that entity in the entire dataset. While third-parties account for 42,592 of the total data flows, only 7 data flows were classified as clear disclosures. As we discuss in Finding 2, this is likely due to the fact that policies are being vague about third-party disclosures.

In contrast, applications were more likely to clearly dis-

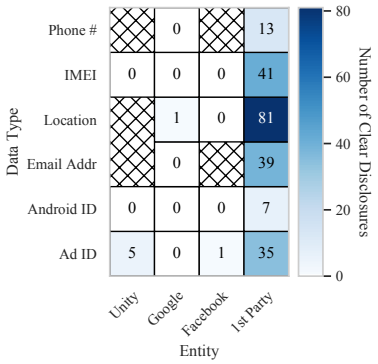

Figure 2: Clear Disclosure HeatMap - Policies are clearer about the first-party receiving a specific data type than a third-party doing so.

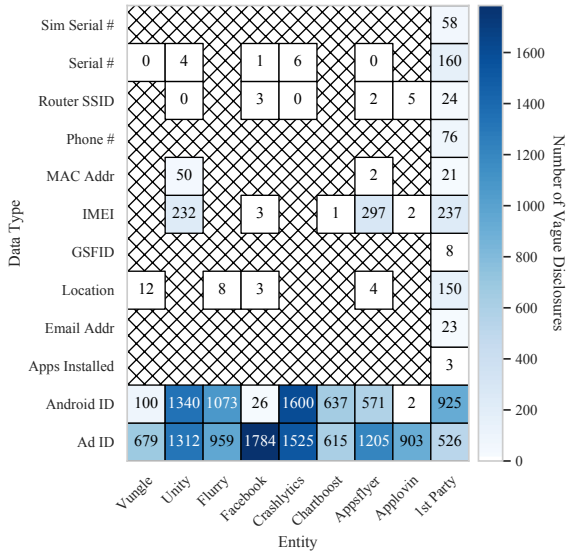

Figure 3: Vague Disclosure Heatmap - Policies often discuss the sharing of Android and advertising IDs in vague terms.

close their first-party data flows (216 data flows across 206 apps). The most commonly disclosed first-party flow involved location data, which was clearly discussed for 81 data flows. However, as there were over 282 instances of first-parties collecting location data, this only accounts for 28.7% (81/ 282) of first-party data flows involving location. Similarly, IMEIs are the second most frequent clear disclosure with 41 data flows, but still only accounted for 12.1% (41/339) of total first-party data flows. The low rate of clear disclosure indicates that privacy policies are not explicitly discussing the types of data that they collect and with whom they share it.

Finding 2: *49.5% of applications are disclosing their third-party sharing practices using vague terms*. In total, 54.9% (23,367/ 42,592) of third-party flows were disclosed using vague terms to refer to the entity, the data type, or both. Figure 3 shows the number of vague disclosures for each of the data flow tuples with the 8 most common third-party entities. The hatched sections denote that there were no transmissions

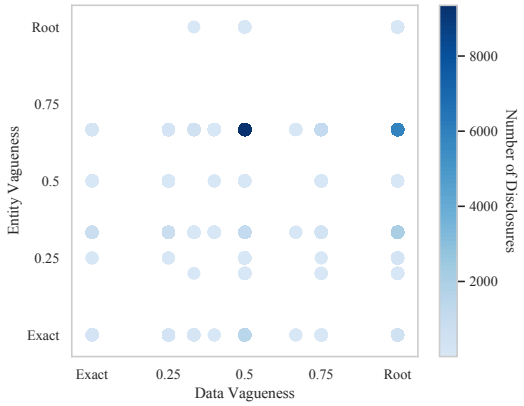

Figure 4: Vagueness Score Scatter Plot - Policies are likely to use vague terms to describe both data types and entities.

of that specific data type to that entity in the entire dataset. Ad IDs and Android IDs accounted for 50.2% (21,363/ 42,592) of the vague disclosures for third-party flows. Ad IDs and Android IDs were disclosed 40.7% of the time by the policy statement (third-party, collect, personally identifiable information) and 25.2% of the time by (third-party, collect, information).

The vagueness of these policy statements does not provide transparency to the wide-range of advertisers and analytics providers that this information is being sent to. As shown in Figure 3, Crashlytics and Unity3d were the most frequent entities of data flows that were classified as vague disclosures. Crashlytics is an analytics provider owned by Google and Unity3d provides a game engine to developers, but also provides advertisements and analytics. In particular, data flows to Crashlytics and Unity3d accounted for 7.4% of third-party vague disclosures (3,131/ 42,592). These entities were discussed as third-parties in 80.7% (2,528/ 3,131) and 72.9% (2,142/ 2,938) of the time, respectively.

Figure 4 shows a graphical representation of the frequency of policy consistency vagueness. Note the root node of the data vagueness is the term "information" while the root node of the entity vagueness is "anyone." The data vagueness score of around 0.5 generally represents terms such as "personally identifiable information," "device information," or "user information." The entity vagueness score of around 0.67 generally represents the term "third-party" while 0.5 represents terms such as "advertising network" or "analytic provider." Therefore, in general, third-party data flows are most frequently described in vague terms for both entities and data types. As the corners of the figure are relatively sparse, it means that if the policy is discussing the entity vaguely then they are likely discussing the data type vaguely. Further, the fact that "third-party" is the most commonly used term to discuss entities, it raises concerns that applications are not complying to the GDPR's mandate on specificity of disclosures.

Figure 5 shows the CDF of the number of unique entities and data types involved within third-party vague disclosures per application. In total, around 80% of the applications with

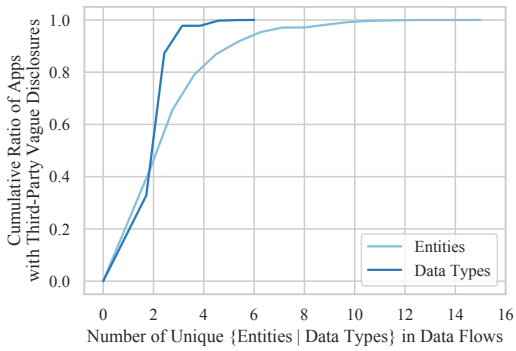

Figure 5: It is feasible for developers to convert vague disclosures to clear disclosures.

third-party vague disclosures contain 4 or fewer unique entities within its data flows. Further, 97.8% of applications with third-party vague disclosures contain 3 or fewer unique data types within its data flows. Therefore, it is largely feasible that developers explicitly disclose the exact data types being shared with the exact entities (clear disclosures) As the developers disclosed the behaviors within the privacy policy, albeit vaguely, they are likely aware that the third-party libraries collect some data. However, it is unknown whether developers are using vague terminology due to not understanding the scope of data collection or whether they do not understand the importance of clear disclosures. Determining the root cause for vague disclosures is left as future work.

Finding 3: *11.6% of applications are disclosing their first-party collection practices using broad terms*. In total, 73.4% (2,211/ 3,011) of first-party flows were disclosed using vague terms to refer to the data type. The right column in Figure 3 shows the distribution of vague disclosures for first-parties. Android IDs accounted for 41.8% (925/ 2,211) of the first-party vague disclosures. Similar to the case for third-parties, these flows were most commonly disclosed as the policy tuple (we, collect, personally identifiable information). Surprisingly, they were only disclosed by the terms "device identifiers" or "identifiers" in 20.8% of the flows (192 / 925). A similar trend follows for first-party collection of Ad IDs and IMEIs. In total, 97.7% of the applications with first-party vague disclosures contain 3 or fewer unique data types within its data flows. Therefore, it is also feasible that developers explicitly disclose the exact data types that they collect (i.e., clear disclosures).

Finding 4: *719 applications make incorrect statements about their data practices*. PoliCheck identified that 719 applications contained incorrect disclosures. These applications consisted of 4.2% (1,930/ 45,603) of the data flows. Figure 6 shows that the most frequent incorrect disclosures involved sharing Ad IDs and Android IDs with Crashlytics (15.7%: 303/ 1,930), Unity3d (13.7%: 264/ 1,930), and Flurry (9.6%: 185/ 1,930). The policy statement (third-party, not_collect, personally identifiable information) accounted for 63.4% disclosures for these cases.

Finding 5: PoliCheck *identified 31.1% (14,409/45,603)*

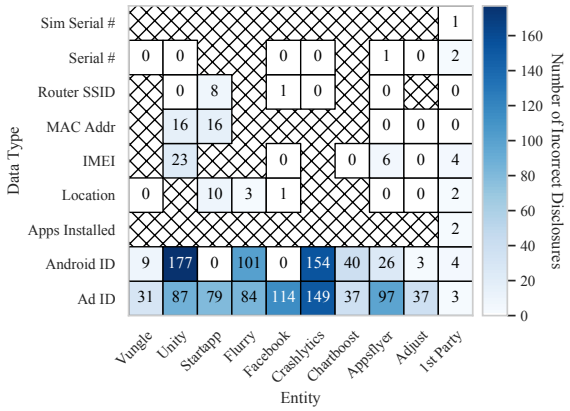

Figure 6: Incorrect Disclosure Heatmap - PoliCheck identified 1,912 incorrect disclosures across 719 applications

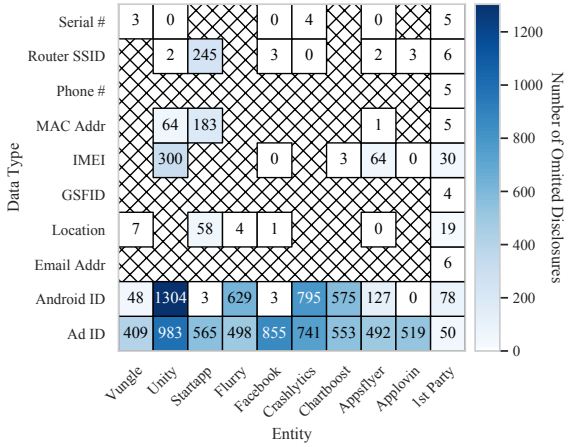

Figure 7: Omitted Disclosure Heatmap - Applications often do not disclose sharing Android and Ad IDs with third-parties. This may be due to the perception that such collection is implied when they disclose that they use an advertiser.

of data flows as omitted disclosures. Of the 14,409 omitted disclosures, 208 were first-party flows and 14,201 involved third-parties. As shown in Figure 7 only 6.9% (208 / 3,011) of first-party flows were not disclosed. The 3 most frequently omitted data types for first-party flows were Android IDs (78/208), Ad IDs (50/208), and the device IMEI (30/208).

For third-party flows, Figure 7 shows that sharing both Android IDs and Ad IDs with Crashlytics and Unity3d accounted for 27.8% (3,168/11,398) and 24.7% (2,810/11,398) omitted disclosures, respectively. Further, sharing AD IDs with Facebook accounts for 15.8% (1,798/11,398) of third-party omitted disclosures. It is surprising that Crashlytics collects Android IDs, as they are a subsidiary of Google, and using persistent hardware identifiers is against Google's outline for the best practices on collecting unique identifiers.

The significant number of omitted disclosures raises the following two questions. First, do developers understand the types of data that are actually being collected when they in-

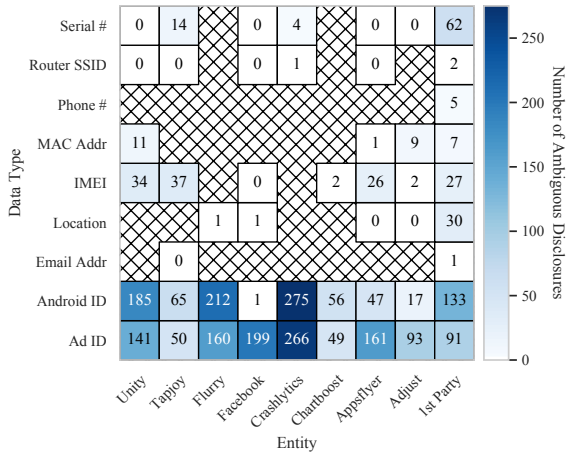

Figure 8: Ambiguous Disclosure Heatmap - Privacy policies are often contradictory when they discuss the sharing of Android and advertising IDs.

clude a third-party SDK into their application? Second, do developers know that they are responsible for disclosing such behaviors in their privacy policy? We leave the exploration of these questions as future work. Note that in comparison to other disclosure types, POLICHECK has lower precision for detecting omitted disclosures. We discuss this in detail in Section 5.2 and provide a case study that they may also be used as indicators for policies that are difficult to comprehend.

Finding 6: *7.6% of applications have ambiguous privacy policies*. In total, 7.6% (3,463/ 45,603) of data flows were classified as ambiguous disclosures, which occurred across 7.6% (1,101/ 13,796) of applications. As shown in Figure 8, Android IDs and Ad IDs are involved in 88.8% (3,074/ 1,101) of ambiguous disclosures. In total, C_1 contradictions were the most common type whose policy statements both state that they do and do not collect information at the same semantic granularity, which accounted for 1,618 types of ambiguous disclosures. For example, on such example is a children's application called "MiraPhone - Kids Phone 4-in-1 apk" (`com.gokids.chydofon`). This application collects the user's Android ID, but the privacy policy explicitly states, "We DO NOT collect your unique identificator [sic]," and also states "Anonymous identifiers, we use anonymous identifiers when you interact with services, such as advertising services and others." These two statements are contradictory policy statements and it is unclear what the correct interpretation of the policy should be.

Finding 7: *Only 2.7% of applications may not be sharing data with third-parties*. In total, only 4.5% (622/13,796) of applications in our dataset did not contain any data flows to third-parties in the observed client-side behavior. 60.0% (373/622) of those applications did not contain statements that disclosed that data may be shared with third-parties. Therefore, assuming that their privacy policy accurately reflects server-side behaviors, only around 2.7% (373/13,796) of applications are not sharing data with third-parties. While the other 40% (249/622) of applications did not contain any data flows to third-parties in the observed client-side behavior, their privacy policy contained statements that disclosed potential third-party sharing. This property may indicate that the data collected within first-party flows may flow to third-parties server-side. For example, we found that 35 applications contained first-party flows collecting a wide-range of data (e.g., location (22 apps), phone number (6 apps), email address (2 apps), applications installed (1 app), Ad IDs (3 apps), and various identifiers (3)). Their privacy policy states that they share data that subsumes the data from a first-party flow to "third-parties," which permits server-side sharing of such data.

5.2 Evaluation

In this section, we present our evaluation of POLICHECK and additional findings from our evaluation. First, we manually validate a random selection of 153 data flows across 151 applications and show that POLICHECK has a 90.8% precision. Second, we perform a sensitivity analysis on POLICHECK's consistency model and show that POLICHECK's entity-sensitive model vastly outperforms entity-insensitive models. The remainder of this section describes these experiments.

5.2.1 POLICHECK's Performance

To evaluate the precision of POLICHECK, we manually validate a subset of data flows. Note that we do not evaluate ambiguous disclosures, as attempting to resolve ambiguity injects annotator bias into the evaluation. For the remaining disclosure types, we randomly select up to 5 data flows for each data type, such that the first-party and third-party data flows are proportionate to the disclosure type's population. Our dataset consists of 180 data flows across 166 apps.

Validation Methodology: For validation, one-of-three authors began by reading through the sentences that were extracted from each privacy policy to ensure correctness of policy statement extraction. If there was an error with policy statement extraction, we record the disclosure as a false positive and stopped analysis. For clear disclosures, vague disclosures, and incorrect disclosures, we locate the sentences in the policy and ensure that the sentence retains the same meaning in the context of the rest of the policy. For omitted disclosures, we read through the rest of the policy to determine if any statements disclose the data flow. If it is not apparent and there is any uncertainty, we mark the flow as "uncertain" to avoid bias. Note that we marked 27 flows as uncertain, resulting in 153 data flows across 151 applications

Results: POLICHECK achieves an overall 90.8% precision (139/153) for performing flow-to-policy consistency analysis. For identifying consistencies (i.e., clear disclosures and vague disclosures), POLICHECK had 86 true positives and only 4 false positives. For incorrect disclosures, POLICHECK had 35

true positives and 3 false positives.

For omitted disclosures, POLICHECK had 18 true positives and 7 false positives, which was primarily due to incomplete policy statement extraction. The main reason for incomplete policy extraction was that the information describing the sharing and collection practices spanned multiple sentences and sections of the policy. The policy did not make declarative statements on their collection and sharing practices. Understanding an entire document is beyond the current limits of NLP, but this stratification also leads to an important observation. The policies for the omitted disclosure false positives were generally more difficult to read than other policies, and often required a great deal of mental effort to understand. Therefore, these omissions can potentially be indicative of poor privacy policy interpretability. We explored this direction by analyzing a select number of applications with the greatest number of omitted disclosures in our data set.

Case Study: Omitted disclosures may also indicate confusing language in privacy policies: A popular game application with over 100M+ downloads called 'Ant Smasher by Best Cool & Fun Games," (`com.bestcoolfungames.antsmasher`) had 17 unique omitted disclosures. The application has an E rating, which means that it is marketed towards children, but yet it shares Ad IDs, Android IDs, and location data with advertisers and analytics providers. When validating these data flows, we found the following policy statement, which potentially discloses these practices albeit vaguely.

> *For instance, whenever you access and start to interact with our Apps, we are able to identify your IP address, system configuration, browser type and other sorts of information arising from your device. We may aggregate that data in order to improve our Apps and other services we provide, but we will not exploit it commercially or disclose it without your consent, except for third-party service providers in order to enable the existence of our Apps and the provision of our services.*

First, they never explicitly mention the data types, but it could arguably fall under the vague umbrella phrase, "other sorts of information arising from your device." Second, the language is unclear and potentially deceptive, because the policy initially implies that device data is not sent to third-parties for commercial reasons. However, it adds an exception for enabling the "existence of their application," which may be interpreted as the revenue from selling user data to advertisers. While our POLICHECK classified this policy as containing omitted disclosures, it is unclear whether this is actually the case and requires analysis by a legal expert. The language that this policy uses is significantly difficult to interpret and that its behaviors should be disclosed more clearly to end-users.

5.2.2 Sensitivity Analysis

To measure the impact of entities in flow-to-policy consistency, we simulated the error rate of entity-insensitive consistency models (i.e., models that do not consider entities) by running consistency analysis in the following three configurations: (1) without entities and without negations (negation-insensitive and entity-insensitive); (2) without entities (entity-insensitive); and (3) without negations (negation-insensitive). Based on the output of the entity-insensitive consistency analysis, we aim to measure the potential error rate. First, we measure the frequency in which third-party data flows are reasoned over using policy statements with semantically unrelated entities (i.e., $f.e \not\sqsubseteq_\varepsilon p.e$). This first metric measures when unrelated policy statements are used to reason whether a flow is consistent. Second, we measure the frequency in which third-party data flows are classified as consistent in the entity-insensitive consistency models that would have been classified as consistent in the inconsistent in entity-sensitive consistency models. This second metric measures when unrelated policy statements cause entity-insensitive models to falsely claim that a data flow is consistent when it is in fact inconsistent. Further, we also measure how these different configurations of consistency models impact the classification of disclosure types.

Finding 8: *Prior entity insensitive flow-to-policy consistency models may wrongly classify up to 37.1% of inconsistent third-party flows as consistent*. We first ran analysis simulating negation-insensitive and entity-insensitive consistency models, such as Slavin et al. [29] and Wang et al. [32]'s models. We found that 53.9% (22,959/ 42,592) of third-party flows were falsely resolved to policy statements that discuss semantically unrelated entities. Of those resolved statements, 39.8% (16,931/ 42,592) referenced first-parties and 14.2% (6,028/ 42,592) references a semantically unrelated third-party. In terms of consistency, 37.1% (15,807/ 42,592) of third-party flows were falsely marked as consistent across 38.4% (5,304/ 13,796) of applications. Of those results, 23.0% (9,779/ 42,592) were due to first-party policy statements and 14.2% (6,028/ 42,592) due to third-party policy statements with a semantically unrelated entity. Therefore, negation-insensitive and entity-insensitive models falsely mark 23.0% inconsistencies as consistent.

We next ran consistency analysis simulation entity-insensitive consistency models, such as Zimmeck et al. [38]. We found 55.8% (23,775/ 42,592) of third-party flows were improperly resolved to policy statements that discuss semantically unrelated entities. Of those resolved statements, 41.6% (17,698/ 42,592) resolved to policy statements referencing first-parties and 14.3% (6,077/ 42,592) resolved to third-parties. In terms of consistency, 30.5% (13,014/ 42,592) of third-party data flows are falsely marked as consistent across 32.2% (4,445/ 13,796) of applications. Of those results, 16.3% (6,937/ 42,592) were due to first-party policy

Table 4: Sensitivity Analysis of Flow-to-Policy Consistency: Entity-insensitive models frequently misclassify data flows

	PoliCheck	(-, -, d)		(-, c, d)		(e, -, d)	
		✓	X	✓	X	✓	X
Clear	223	223	3,180	216	1,856	223	39
Vague	25,578	22,964	17,149	18,122	9,852	25,578	5,354
Omitted	14,409	2,087	0	2,087	0	14,409	0
Incorrect	1,930	0	0	558	5,081	0	0
Ambiguous	3,463	0	0	2,298	5,533	0	0

** (-, -, d): entity-insensitive and negation-insensitive*
** (-, c, d): entity-insensitive and negation-sensitive*
** (e, -, d): entity-sensitive and negation-insensitive*

statements and 14.3% (6,077/ 42,592) due to third-party policy statements with a semantically unrelated entity. Therefore, entity-insensitive models falsely mark 30.5% of inconsistencies as consistent.

Finding 9: *Entity-insensitive analysis results in the frequent misclassification of disclosure types*. Table 4 shows the results of our sensitivity analysis for classifying each disclosure type. Overall, entity-insensitive consistency models have the worst performance at classifying disclosure types, as they significantly overestimate the number of clear disclosures and vague disclosures. Negation-insensitive consistency models cannot detect incorrect disclosures or ambiguous disclosures, which correspond to 4.2% and 7.6% of data flows, respectively. With negation-insensitive consistency models, the incorrect disclosures or ambiguous disclosures are wrongly classified as either clear disclosures or vague disclosures, which is concerning as these models would state that the data flow is consistent with the policy. While consistency models that are negation-sensitive and entity-insensitive *(-, c, d)* can theoretically identify incorrect disclosures and ambiguous disclosures, the results show that their identification of these disclosure types are imprecise due to not considering entities. The results from this analysis demonstrate both the importance of entity-sensitive and negation-sensitive analysis at classifying disclosure types and the unprecedented view that PoliCheck's flow-to-policy consistency model provides on privacy disclosures.

6 Additional Case Studies

The examples in Section 2 provide real-world case-studies that demonstrate PoliCheck's utility, the significance of our findings, and the importance of an entity-sensitive consistency model. In this section, we provide additional case studies from our analysis of the most inconsistent applications for each consistency type (i.e., the applications in the long tail in Figure 9). We analyzed each data flow, the policy statements extracted, and the privacy policy itself to validate the findings. The remainder of this section presents concrete examples.

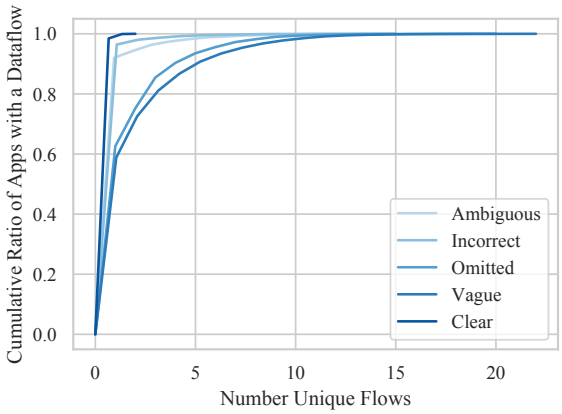

Figure 9: The majority of applications have less than five data flows for each disclosure type, but a small percentage have significantly more.

6.1 Omitted Disclosures

We investigated applications in our dataset with high numbers of omitted data flows. "Survival Island Games - Survivor Craft Adventure" (`com.gamefirst.chibisurvivor`) is a game with over 500K installs on Google play. We found that the app collects the user's location data, Android ID, and MAC address to share with advertisers and analytics providers. Its privacy policy does not discuss any details regarding data sharing. Omitted disclosures are grave concerns, especially in cases like the one above, which involves tracking the user's physical location along with persistent identifiers.

While validating omitted disclosures, we found another application called "Cloudventure: Arcade + Editor" (`at.hakkon.pufpuf.android`) that has an omitted disclosure of Ad ID being shared with AdColony. The privacy policy is copied below in entirety, which shows the potential deceptiveness of their policy.

> *Okay guys listen up, I'm forced to write this privacy policy or Google will take this APP from the store. - There is an option in the app to share your level with your friends. This is made by making a screenshot of your screen and is the reason why camera permission is needed.*

- Also this is a game so you don't want the screen to go dark while playing, right? That's why I need the phone state permission. (android.permission.READ_PHONE_STATE)

That's it, this app is not evil and I'm not selling your data to some crazy marketing company to get you filled up with spam.

6.2 Incorrect Disclosures

Three of the top five applications that had the greatest number of incorrect disclosures were released by the same publisher, "Nazara Games." Their games on Google Play have over 57 million total downloads for 33 applications (eight of which occur in our dataset). They publish games with an E rating, which may be used by targeted towards children, but still collect a wide-array of privacy-sensitive data. From the 8 applications in our dataset, we found 95 flows originating from Nazara Games applications, of which, 75 had incorrect disclosures. Their application "Chhota Bheem Speed Racing" (`com.nazara.tinylabproductions.chhotabheem-22002`) has over 10M+ downloads and has 15 incorrect disclosures detected by PoliCheck. These flows included location data, Android IDs, Ad IDs, IMEIs, router SSIDs, and other serial numbers. Nazara Games' applications sent this data to 14 distinct advertisers and analytics providers, such as Flurry, ironSource, and Unity3d Ads. As some of their applications targeted towards children, the mass collection and sharing of this sensitive user data is egregious. Even more so when considering that they're sending this information is likely being used to target ads towards children.

In Nazara Games' privacy policies, that they do not sell or rent personal information unless the user gives consent.

> *Nazara does not sell or rent your Personal Information to third-parties for marketing purposes without your consent.*

As some of these applications are for children, verifiable consent is required from the child's legal guardian according to regulations [2]. As discussed by prior work [27], clicking a button likely does not constitute verifiable consent. For the applications that are not targeted towards children, it is unclear if consent is explicitly request or implicitly through acceptance of the policy. We leave it as future work to analyze how applications are requesting consent.

6.3 Ambiguous Disclosures

"Roller Coaster Tycoon 4" (`com.atari.mobile.rct4m`) is a popular game from Atari which has over 10M downloads. We found that this application has 15 ambiguous disclosures due to their sharing of Ad IDs, Android IDs, and IMEI with advertisers and analytics providers, such as TapJoy, ironSource, and AdColony. Atari does not consider device information to be PII. However, various regulations [1, 3] identify such information as PII, as they can be used to identify users over a long span of time across different applications and services. The main source of ambiguous disclosures were due to statements regarding allowing business partners to collect device identifiers, but then stating that third-parties will not collect device identifiers without consent.

The "Bowmasters" game application (`com.miniclip.bowmasters`) has over 50M downloads and 12 unique ambiguous disclosures. Their policy states "We don't give or sell your data to third-parties for them to market to you", but later it states, "On our apps, these third-party advertising companies will collect and use your data to provide you with targeted advertising." As serving targeted advertisements is a form of marketing, this policy contradicts itself and is ambiguous in terms of the flow.

7 Limitations

PoliCheck provides a concise formalization of an entity-sensitive flow-to-policy consistency model and disclosure types. Our findings from Section 2, Section 5, and Section 6 demonstrate the utility and value of such analysis. However, as the current implementation of PoliCheck is built on top of PolicyLint [4] and AppCensus [6], we inherit their limitations. For example, PolicyLint's performance depends on the completeness of the verb lists a policy statement patterns, which may impact overall recall. PolicyLint also does not extract the purpose of collection, which we leave as future work. Further, the data flows used by PoliCheck may also be incomplete if the behaviors were not executed during runtime due to lack of code coverage. In addition, PoliCheck only tracks the data types in Table 2. Future work can improve completeness of policy statement extraction and dynamic analysis, which can then be used as input to PoliCheck.

Another limitation is that PoliCheck's domain-to-entity mapping may be incomplete, as our study is primarily focused on popular advertisers and analytics providers. PoliCheck's approach for classifying first-party entities also has the potential for misclassifying third-party flows as first-parties if the privacy policies are hosted on third-party domains. However, misclassification would also require a data flow to that domain within the application, which was not observed during validation. Additional techniques are also required for resolving cloud hosts and content-delivery networks to entities, such as Razaghpanah et al.'s certificate-based approach [23]. As discussed in Section 5, we discard data flows where the entity could not be resolved. Therefore, a more comprehensive mapping and resolution will improve the completeness of our analysis but will not impact the soundness of our empirical study in terms of the classification of disclosure types. Future work can explore more comprehensive approaches for resolving domains and IP addresses to entities and constructing domain-to-entity mappings.

Moreover, while PoliCheck correctly reasons over third-party disclosures that are disclosed in terms of parent companies (i.e., subsidiary relationships), the current implementation does not capture subsidiary relationships of first-party disclosures. While we did not observe this limitation resulting in false positives during validation, future work can adapt the entity ontology based on the application under analysis to address this limitation.

Finally, our empirical study focuses on the privacy policies of Android applications. While we cannot claim that our findings generalize to other platforms (e.g., iOS, web), we hypothesize that our findings on the disclosure types would likely mirror other domains, as the policies are generally written to cover multi-platform applications and similar data types are available for collection in other platforms.

8 Related Work

In recent years, there has been an increased focus on analyzing flow-to-policy inconsistencies in mobile applications. The works differ in how app behavioral flows and privacy policies are analyzed. While much of the prior works [29, 34, 38] use Android's application program interface (API) calls to evaluate privacy breaches, Wang et al. [32] extended the taint sources to include sensitive data entered through an app's UI. For policy analysis, Zimmeck et al. [38] and Yu et al. [34] rely on keyword-based approaches, of using bi-grams and verb modifiers respectively, to infer the privacy policies, while Slavin et al. [29] and Wang et al. [32] use crowdsourced ontologies for policy analysis. PoliCheck makes significant advancement over all these prior works by considering DNS domains of data-receiving entity for comprehensive entity-sensitive analysis. Accuracy of the analysis is further improved by considering entities, statement sentiment, and accounting for different semantic granularities and internal contradictions. Our empirical results (Section 5) further demonstrate the effectiveness of these capabilities.

Other recent research has focused on analyzing specific application categories, such as those designed for families, for compliance and privacy violations [17, 21, 26, 27]. Similar to PoliCheck, they use dynamic analysis to identify sensitive flows along with the entities receiving the data. However, their policy analysis is either manual [17, 26, 27] or semi-automatic based on keyword searches [21]. While these approaches potentially worked for a category of apps with explicit requirements, they are severely limited in precision and scale for broader categories as targeted by our work. In contrast, PoliCheck uses an automated, comprehensive policy analysis that improves precision by considering additional capabilities, such as semantic granularities and contradictions.

Numerous works focus on the automated analysis of privacy policies themselves. Privee [37] uses natural language processing for deriving answers to a limited set of binary questions from the privacy policies, while Hermes [31] applies topic modeling to reduce ambiguity in privacy policies. PrivacyCheck [35] use data mining models to analyze the privacy policies to automatically extract their graphical summaries representing what information is used and how. A more recent work, Polisis [19], provides an automated policy analysis tool that uses deep learning to infer types of data collected and the reason for collection. While it provides alternate approaches for comprehensive policy analysis, it does not consider negations and exclusions in text. PolicyLint [4] recently showed that a considerable number of policies include negations and exclusions that would be missed by prior works. Our policy analysis is built on top of PolicyLint and hence improves precision over prior art. Moreover, none of the works focus on the evaluation of app behavior, which is a core component for our entity-sensitive flow-to-policy analysis.

There is a rich body of work to understand [8, 13, 22, 33] and bridge [28, 36] the gap between application behaviors and users' understanding of these behaviors. PoliCheck differs from these works in its focus of analyzing privacy policy to behavior inconsistencies.

9 Conclusion

Privacy threats from mobile applications are arguably a greater risk than malware for most smartphone users. While the last decade has produced many static and dynamic analysis to detect when mobile applications send privacy-sensitive data to the network, such data flows are not privacy leaks if they are disclosed in a privacy policy. Recently, several efforts have sought to more fully automate the detection of privacy leaks by contrasting data flows with the application's privacy policy. However, these works have a fundamental limitation: they do not consider the entity receiving the data (e.g., first-party vs. third-party). In this paper, we proposed PoliCheck and an entity-sensitive flow-to-policy consistency model. We used PoliCheck to study 13,796 applications, comparing their data flows to their policy statements. We find significant evidence of omitted, incorrect, and ambiguous disclosures, many of which are only possible to identify by considering the entity. As such, PoliCheck provides the highest-precision method to date to determine if apps properly disclose their privacy-sensitive behaviors.

Acknowledgment

We thank our shepherd, Anita Nikolich, and the anonymous reviewers for their valuable comments. This work is supported in part by NSF grant CNS-1513690. Any findings and opinions expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] California Consumer Privacy Act (CCPA). https://oag.ca.gov/privacy/ccpa.

[2] Children's Online Privacy Protection Rule. https://www.ftc.gov/enforcement/rules/rulemaking-regulatory-reform-proceedings/childrens-online-privacy-protection-rule.

[3] The EU General Data Protection Regulation. https://eugdpr.org.

[4] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play. In *Proceedings of the USENIX Security Symposium*, August 2019.

[5] Android Studio. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/monkey.html, 2019. Accessed: May 15, 2019.

[6] AppCensus AppSearch. https://search.appcensus.io/.

[7] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. In *Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI)*, 2014.

[8] David Barrera, H. Güneş Kayacik, Paul C. van Oorschot, and Anil Somayaji. A Methodology for Empirical Analysis of Permission-based Security Models and Its Application to Android. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, October 2010.

[9] J. Bowers, B. Reaves, I. Sherman, P. Traynor, and K. Butler. Regulators, Mount Up! Analysis of Privacy Policies for Mobile Money Applications. In *Proceedings of the USENIX Symposium on Usable Privacy and Security (SOUPS)*, 2017.

[10] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. PiOS: Detecting Privacy Leaks in iOS Applications. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, February 2011.

[11] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 2010.

[12] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A Study of Android Application Security. In *Proceedings of the USENIX Security Symposium*, August 2011.

[13] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android Permissions Demystified. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, October 2011.

[14] Xinming Ou Fengguo Wei, Sankardas Roy and Robby. Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, November 2014.

[15] In the Matter of Goldenshores Technologies, LLC, and Erik M. Geidl. https://www.ftc.gov/enforcement/cases-proceedings/132-3087/goldenshores-technologies-llc-erik-m-geidl-matter.

[16] Michael Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. Unsafe Exposure Analysis of Mobile In-App Advertisements. In *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2012.

[17] Catherine Han, Irwin Reyes, Amit Elazari Bar On, Joel Reardon, Álvaro Feal, Kenneth A. Bamberger, Serge Egelman, and Narseo Vallina-Rodriguez. Do You Get What You Pay For? Comparing The Privacy Behaviors of Free vs. Paid Apps. In *Workshop on Technology and Consumer Protection (ConPro)*, May 2019.

[18] Jin Han, Qiang Yan, Debin Gao, Jianying Zhou, and Robert Deng. Comparing Mobile Privacy Protection through Cross-Platform Applications. In *Proceedings of the ISOC Network and Distributed Systems Symposium (NDSS)*, February 2013.

[19] Hamza Harkous, Kassem Fawaz, Rémi Lebret, Florian Schaub, Kang G. Shin, and Karl Aberer. Polisis: Automated Analysis and Presentation of Privacy Policies Using Deep Learning. In *Proceedings of the USENIX Security Symposium*, 2018.

[20] K. Butler J. Bowers, I. Sherman and P. Traynor. Characterizing Security and Privacy Practices in Emerging Digital Credit Applications. In *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2019.

[21] Ehimare Okoyomon, Nikita Samarin, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, Irwin Reyes, Álvaro Feal, and Serge Egelman. On The Ridiculousness of Notice and Consent: Contradictions in App Privacy Policies. In *Workshop on Technology and Consumer Protection (ConPro)*, May 2019.

[22] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Alan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using Probabilistic Generative Models for Ranking Risks of Android Apps. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, October 2012.

[23] Abbas Razaghpanah, Rishab Nithyanand, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Mark Allman, Christian Kreibich, and Phillipa Gill. Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2018.

[24] Joel Reardon, Alvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permission System. In *Proceedings of the USENIX Security Symposium*, 2019.

[25] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David R. Choffnes. ReCon: Revealing and Controlling Privacy Leaks in Mobile Network Traffic. In *Proceedings of the ACM SIGMOBILE MobiSys*, pages 361–374, 2016.

[26] Irwin Reyes, Primal Wiesekera, Abbas Razaghpanah, Joel Reardon, Narseo Vallina-Rodriguez, Serge Egelman, and Christian Kreibich. "Is Our Children's Apps Learning?" Automatically Detecting COPPA Violations. In *Workshop on Technology and Consumer Protection (ConPro)*, May 2017.

[27] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, and Serge Egelman. "Won't Somebody Think of the Children?" Examining COPPA Compliance at Scale. In *Proceedings on Privacy Enhancing Technologies (PETS)*, July 2018.

[28] Sanae Rosen, Zhiyun Qian, and Z. Morely Mao. AppProfiler: A Flexible Method of Exposing Privacy-related Behavior in Android Applications to End Users. In *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY*, February 2013.

[29] Rocky Slavin, Xiaoyin Wang, Mitra Bokaei Hosseini, James Hester, Ram Krishnan, Jaspreet Bhatia, Travis D. Breaux, and Jianwei Niu. Toward a Framework for Detecting Privacy Policy Violations in Android Application Code. In *Proceedings of the International Conference on Software Engineering (ICSE)*, 2016.

[30] In the Matter of Snapchat, Inc. https://www.ftc.gov/enforcement/cases-proceedings/132-3078/snapchat-inc-matter.

[31] John W. Stamey and Ryan A. Rossi. Automatically Identifying Relations in Privacy Policies. In *Proceedings of the ACM International Conference on Design of Communication (SIGDOC)*, 2009.

[32] Xiaoyin Wang, Xue Qin, Mitra Bokaei Hosseini, Rocky Slavin, Travis D. Breaux, and Jianwei Niu. GUILeak: Tracing Privacy Policy Claims on User Input Data for Android Applications. In *Proceedings of the International Conference of Software Engineering (ICSE)*, 2018.

[33] Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. Android Permissions Remystified: A Field Study on Contextual Integrity. In *Proceedings of the USENIX Security Symposium*, August 2015.

[34] Le Yu, Xiapu Luo, Xule Liu, and Tao Zhang. Can We Trust the Privacy Policies of Android Apps? In *Proceedings of the IEEE/IFIP Conference on Dependable Systems and Networks (DSN)*, 2016.

[35] Razieh Nokhbeh Zaeem, Rachel L. German, and K. Suzanne Barber. PrivacyCheck: Automatic Summarization of Privacy Policies Using Data Mining. *ACM Transactions on Internet Technology (TOIT)*, 2013.

[36] Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, X. Sean Wang, and Binyu Zang. Vetting Undesirable Behaviors in Android Apps with Permission Use Analysis. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, November 2013.

[37] Sebastian Zimmeck and Steven M. Bellovin. Privee: An Architecture for Automatically Analyzing Web Privacy Policies. In *Proceedings of the USENIX Security Symposium*, 2014.

[38] Sebastian Zimmeck, Ziqi Wang, Lieyong Zou, Roger Iyengar, Bin Liu, Florian Schaub, Shomir Wilson, Norman Sadeh, Steven M. Bellovin, and Joel Reidenberg. Automated Analysis of Privacy Requirements for Mobile Apps. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, 2017.