PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play

Benjamin Andow and Samin Yaseer Mahmud, *North Carolina State University;* Wenyu Wang, *University of Illinois at Urbana-Champaign;* Justin Whitaker, William Enck, and Bradley Reaves, *North Carolina State University;* Kapil Singh, *IBM T.J. Watson Research Center;* Tao Xie, *University of Illinois at Urbana-Champaign*

https://www.usenix.org/conference/usenixsecurity19/presentation/andow

This paper is included in the Proceedings of the 28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play

Benjamin Andow⋆, Samin Yaseer Mahmud⋆, Wenyu Wang†, Justin Whitaker⋆
William Enck⋆, Bradley Reaves⋆, Kapil Singh‡, Tao Xie†
⋆*North Carolina State University*
†*University of Illinois at Urbana-Champaign*
‡*IBM T.J. Watson Research Center*

Abstract

Privacy policies are the primary mechanism by which companies inform users about data collection and sharing practices. To help users better understand these long and complex legal documents, recent research has proposed tools that summarize collection and sharing. However, these tools have a significant oversight: they do not account for contradictions that may occur within an individual policy. In this paper, we present PolicyLint, a privacy policy analysis tool that identifies such contradictions by simultaneously considering negation and varying semantic levels of data objects and entities. To do so, PolicyLint automatically generates ontologies from a large corpus of privacy policies and uses sentence-level natural language processing to capture both positive and negative statements of data collection and sharing. We use PolicyLint to analyze the policies of 11,430 apps and find that 14.2% of these policies contain contradictions that may be indicative of misleading statements. We manually verify 510 contradictions, identifying concerning trends that include the use of misleading presentation, attempted redefinition of common understandings of terms, conflicts in regulatory definitions (e.g., US and EU), and "laundering" of tracking information facilitated by sharing or collecting data that can be used to derive sensitive information. In doing so, PolicyLint significantly advances automated analysis of privacy policies.

1 Introduction

Mobile apps collect, manage, and transmit some of the most sensitive information that exists about users—including private communications, fine-grained location, and even health measurements. These apps regularly transmit this information to first or third parties [1, 7, 14]. Such data collection/sharing by an app is often considered (legally) acceptable if it is described in the privacy policy for the app. Privacy policies are sophisticated legal documents that are typically long, vague, and difficult for novices, experts, and algorithms to interpret. Accordingly, it is difficult to determine whether app developers adhere to privacy policies, which can help app markets and other analysts identify privacy violations, or help end users choose more-privacy-friendly apps.

Recent work has begun studying whether or not mobile app behavior matches statements in privacy policies [26,28,29,32]. However, the prior work fails to account for contradictions within privacy policies; these contradictions may lead to incorrect interpretation of sharing and collection practices. Identifying contradictions requires overcoming two main challenges. First, privacy policies refer to information at different semantic granularities. For example, a policy may discuss its practices using broad terms (e.g., "personal information") in one place in the policy, but later discuss its practices using more specific terms (e.g., "email address"). Prior approaches have tackled this issue by crowdsourcing data object ontologies [26, 28],[1] but such crowdsourced information is not complete, accurate, or easily collected. Second, prior approaches have struggled to accurately detect negative statements, relying on bi-grams (e.g., "not share") [32] or detecting only verb modifiers [29] while neglecting the more-complicated statements (e.g., "will share X except Y") that are common in privacy policies. Modeling negative statements is required to determine the correct meaning of a policy statement (i.e., "not sharing" versus "sharing" information). Fully characterizing contradictions requires addressing both preceding challenges.

In this paper, we present PolicyLint for automatically identifying potential contradictions of sharing and collection practices in software privacy policies. Contradictions make policies unclear, confusing both humans and any automated system that rely on interpreting the policies. Considering these uses cases, PolicyLint defines two contradiction groupings. *Logical contradictions* are contradictory policy statements that are more likely to cause harm if users and analysts are not aware of the contradictory statements. One example is a policy that initially claims not to collect personal information, but later in fine print discloses collecting a user's name and email address for advertisers. *Narrowing definitions* may cause automated techniques that reason over policy statements

[1] Ontologies are graph data structures that capture relationships among entities. For example, "personal information" subsumes "your email address."

to make incorrect or inconsistent decisions and may result in vague policies. PolicyLint is the first tool to have the sophistication necessary to reason about both negative sentiments and statements covering varying levels of specificity, being necessary for uncovering contradictions.

PolicyLint is inspired by other security `lint` tools [6, 8–11, 19], which analyze code for indicators of potential bugs. Like any static approach, not every `lint` finding is necessarily a real bug. For example, potential bug conditions could be mitigated by an external control or other context that the tool cannot verify. In many cases, only a human can verify the outputs of a `lint` finding. In the case of PolicyLint, we note that privacy policies are complex legal documents that may be intentionally vague, ambiguous, or misleading even for human interpretation. Despite these challenges, PolicyLint condenses long, complicated policies into a small set of candidate issues of interest to human or algorithmic analysis.

This paper makes the following main contributions:

- **Automated generation of ontologies from privacy policies.** PolicyLint uses an expanded set of Hearst patterns [16] to extract ontologies for both data objects and entities from a large corpus of privacy policies (e.g., "W such as X, Y, and Z"). PolicyLint is more comprehensive and scalable than crowdsourced efforts [26, 28].

- **Automated sentence-level extraction of privacy practices.** PolicyLint uses sentence-level NLP and leverages parts-of-speech and type-dependency information to capture data collection and sharing as a four-tuple: (actor, action, data object, entity). For example, "We [actor] share [action] personal information [data object] with advertisers [entity]." Sentence-level NLP is critically important for the correct identification of negative statements. We also show that prior attempts at analyzing negation would fail on 28.2% of policies.

- **Automated analysis of contradictions in privacy policies.** We formally model nine types of contradictions that result from semantic relationships between terms, providing an algorithmic technique to detect contradictory policy statements. Our groupings of *narrowing definitions* and *logical contradictions* lay the foundation for ensuring the soundness of automated policy tools and identifying potentially misleading policy statements. In a study of 11,430 privacy policies from mobile apps, we are the first to find that logical contradictions and narrowing definitions are *rampant*, affecting 17.7% of policies, with logical contradictions affecting 14.2%.

- **Manual analysis of contradictions to identify trends.** The high ratio of policy contradictions is surprising. We manually review 510 contradictions across 260 policies, finding that many contradictions are indeed indicators of misleading or problematic policies. These contradictions include making broad claims to protect personal information early in a policy, yet later carving out exceptions for data that the authors attempt to redefine as not personal, that could be used to derive sensitive information (e.g., IP addresses and location), or that are considered sensitive by some regulators but not others.

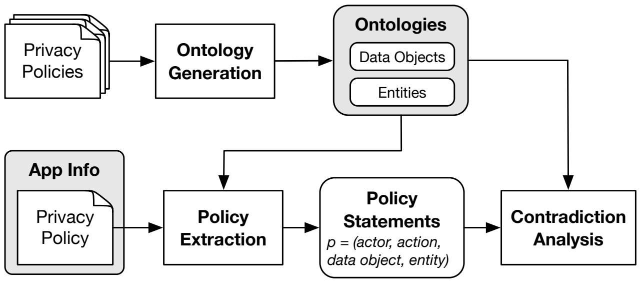


Figure 1: Overview of PolicyLint

PolicyLint has four main potential use cases. First, policy writers can leverage PolicyLint to reduce the risk of releasing misleading policies. In fact, when we contacted parties responsible for the contradictory policies, several fixed their policies (Section 3.4). Second, regulators can use PolicyLint's definition of logical contradictions to identify deceptive policies. While the FTC has identified contradictory statements as problem areas within privacy policies [3], to our knowledge, there is no legal precedent regarding whether regulatory agencies would take action as a result of contradictory policies. However, we believe that some of our findings in Section 3 potentially fall under the FTC's definition of deceptive practices [13]. We envision that regulators could deploy PolicyLint to audit companies' privacy policies for misleading statements at large scale. Third, app markets, such as Google Play, can deploy PolicyLint similarly to ensure that apps posted in the store do not have misleading statements in their privacy policies. Furthermore, they can also use PolicyLint's extraction of policy statements to automatically generate privacy labels to display on the markets to nudge users to less-privacy-invasive apps. Finally, automated techniques for analyzing privacy policies can use PolicyLint's fine-grained extraction of policy statements and formalization of logical contradictions and narrowing definitions to help ensure tool soundness.

The rest of this paper is organized as follows. Section 2 describes PolicyLint's design. Section 3 reports on our study using PolicyLint. Section 4 discusses limitations and future work. Section 5 describes related work. Section 6 concludes.

2 PolicyLint

PolicyLint seeks to identify contradictions within individual privacy policies for software. It provides privacy *warnings* based on contradictory sharing and collection statements within policies, but similar to lint tools for software, these warnings require manual verification. PolicyLint identifies "candidate contradictions" within policies. A candidate contradiction is a pair of contradictory policy statements when considered in the most conservative interpretation (i.e., context-insensitive). Candidate contradictions that are validated by analysts are termed as "validated contradictions." Manual verification is required due to the fundamental problems of ambiguity when interpreting the meaning of natural language sentences (i.e., multiple interpretations of the same sentence).

For example, consider the privacy policy for a popular recipe app (com.omniluxtrade.allrecipes). One part of the policy states "We do not collect personally identifiable information from our users." It is clear from this sentence that the app does not collect any personal information. However, later the policy states, "We may collect your email address in order to send information, respond to inquiries, and other requests or questions." Such sentence is a clear contradiction to the earlier sentence, as email address is considered personal information. As discussed in detail in Section 3, the cause for this underlying contradiction is that the developer does not consider email address as personal information.

To our knowledge, we are the first to characterize and automatically analyze contradictions within privacy policies. While PolicyLint is not the first NLP tool to analyze privacy policies, identifying contradictions requires addressing two broad challenges.

- *References to information are expressed at different semantic levels.* Prior work [26, 28] uses ontologies to capture subsumptive (i.e., "is-a") relationships between terms; however, such ontologies are crowdsourced and subsumptive relationships are manually defined by the authors, leaving concerns of comprehensiveness and scalability. For example, prior work [26, 28] builds their ontology using only 50 and 30 policies, respectively. While crowdsourced ontologies could be comprehensive given unlimited time and manpower, crowdsourcing at large scale is infeasible due to limited resources. Furthermore, existing general-purpose ontologies do not capture all of the specific relationships required to reason over data types and entities mentioned within privacy policies.

- *Privacy policies include negative sharing and collection statements.* Most prior work [26, 28] operates at paragraph level and cannot capture negative sharing statements. Prior work [29, 32] that does capture negative statements misses complex statements (e.g., "will share personal information except your email address"). Such prior work extracts coarse-grained summaries of policy statements (paragraph-level [26, 28], document-level [32]) and can never precisely model negative statements or the entities involved. Their imprecision may result in incorrectly reasoning about 28.2% of policies due to their negation modeling (Finding 1 in Section 3).

We tackle these challenges using two key insights.

Sentence structure informs semantics: Sharing and collection statements generally follow a learnable set of templates. PolicyLint uses these templates to extract a four tuple from such statements: (actor, action, data object, entity). For example, "We [actor] share [action] personal information [data object] with advertisers [entity]." The sentence structure also provides greater insight into more complex negative sharing. For example, "We share personal information except your email address with advertisers." PolicyLint extracts such semantics from policy statements by building on top of existing parts-of-speech and dependency parsers.

Privacy policies encode ontologies: Due to the legal nature of privacy policies, general terms are often defined in terms of examples or their constituent parts. While each policy might not define semantic relationships for all of the terms used in the policy, those relationships should exist in some other policies in our dataset. By processing a large number of privacy policies, PolicyLint automatically generates an ontology specific to policies (one for data objects and one for entities). PolicyLint extracts term definitions using Hearst patterns [16], which we have extended for our domain.

Figure 1 depicts the data flow within PolicyLint. There are three main components of PolicyLint: ontology generation, policy extraction, and contradiction analysis. The following sections describe these components. Readers interested in policy-preprocessing considerations can refer to Appendix A.

2.1 Ontology Generation

The goal of ontology generation is to define subsumptive ("is-a") relationships between terms in privacy policies to allow reasoning over different granularities of language. PolicyLint operates on the intuition that subsumptive relationships are often embedded within the text of a privacy policy, e.g., an example of the types of data considered to be a specific class of information. The following example identifies that demographic information subsumes age and gender.

Example 1. *We may share demographic information, such as your age and gender, with advertisers.*

PolicyLint uses such sentences to automatically discover subsumptive relationships across a large set of privacy policies. It focuses on data objects and the entities receiving data.

PolicyLint uses a semi-automated and data-driven technique for ontology generation. It breaks ontology generation into three main parts. First, PolicyLint performs domain adaptation of an existing model of statistical-based named entity recognition (NER). NER is used to label data objects and entities within sentences, capturing not only terms, but also surrounding context in the sentence. Second, PolicyLint learns subsumptive relationships for labeled data objects and entities by using a set of 11 lexicosyntactic patterns with enforced named-entity label constraints. Third, PolicyLint takes a set

Table 1: NER Performance: Comparison of spaCy's stock en_core_web_lg model versus our adapted domain model

	Overall		Data Objects		Entities	
	Default	Adapted	Default	Adapted	Default	Adapted
Precision	43.48%	84.12%	-	82.20%	61.22%	86.75%
Recall	8.33%	81.67%	-	79.84%	17.75%	85.21%
F1-Score	13.99%	82.88%	-	81.00%	27.52%	85.97%

of seed words as input and generates data-object/entity ontologies using the subsumptive relationships discovered in the prior step. It iteratively adds relationships to the ontology until a fixed point is reached. We next describe this process.

2.1.1 NER Domain Adaptation

To identify subsumptive relationships for data objects and entities, PolicyLint must identify which sentence tokens represent a data object or entity. For Example 1, we seek to identify "demographic information," "age," and "gender" as data objects, and "we" and "advertisers" as entities. PolicyLint uses a statistical-based technique of named-entity recognition (NER) to label data objects and entities within sentences. Prior research [26, 28, 29, 32] proposed keyphrase-based techniques for identifying data objects. However, keyphrase-based techniques are less versatile in practice: they cannot handle term ambiguity and variability, and they can identify only terms in their pre-defined list. For example, "internet service provider" can be both a data object and entity, which keyphrase-based techniques cannot differentiate. In contrast, statistical-based NER both resolves ambiguity and discovers "unseen" terms.

Unfortunately, existing NER models are not trained for our problem domain (data objects and collective terms describing entities, e.g., "advertisers"). Training an NER model from scratch is time-consuming due to the large amount of training data required to achieve reasonable performance. Therefore, PolicyLint takes an existing NER model and updates it using annotated training data from our problem domain. Specifically, PolicyLint adopts spaCy's NER engine [17], which uses deep convolutional neural networks. We adapt the *en_core_web_lg* model to the privacy policy domain.

To perform domain adaptation, we gather 500 sentences as training data. Our training data is selected as follows. First, we randomly select 50 unique sentences from our policy dataset. Second, for each of the 9 lexicosyntactic patterns described in Section 2.1.2, we randomly select 50 sentences that contain the pattern (450 in total). We run the existing NER model on the training sentences to prevent the model from "forgetting" old annotations. We then manually annotate the sentences with data objects and entities.

When updating the existing NER model, we perform multiple passes over the annotated training data, shuffling at each epoch, and using minibatch training with a batch size of 4. To perform the domain adaptation, the current model predicts the NER labels for each word in the sentence and adjusts the synaptic weights in the neural network accordingly if the prediction does not match the annotation. We stop making passes over the training data when the loss rate begins to converge. We annotate an additional 100 randomly selected sentences as holdout data for testing the model. Table 1 shows the NER model performance before and after domain adaptation for our holdout dataset. PolicyLint achieves 82.2% and 86.8% precision for identifying data objects and entities, respectively.

Table 2: Lexicosyntatic patterns for subsumptive relationships

#	Pattern
H1	X, *such as* $Y_1, Y_2, \ldots, Y_n$
H2	*such* X *as* $Y_1, Y_2, \ldots Y_n$
H3	X $\left[\textit{or}\vert\textit{and}\right]$ *other* $Y_1, Y_2, \ldots Y_n$
H4	X, *including* $Y_1, Y_2, \ldots Y_n$
H5	X, *especially* $Y_1, Y_2, \ldots Y_n$
H′1	X, $\left[\textit{e.g.}\vert\textit{i.e.}\right]$, $Y_1, Y_2, \ldots Y_n$
H′2	X ($\left[\textit{e.g.}\vert\textit{i.e.}\right]$, $Y_1, Y_2, \ldots Y_n$)
H′3	X, *for example*, $Y_1, Y_2, \ldots Y_n$
H′4	X, *which may include* $Y_1, Y_2, \ldots Y_n$

* H* = Hearst Pattern; H′* = Custom Pattern

2.1.2 Subsumptive Relationship Extraction

PolicyLint uses a set of 9 lexicosyntactic patterns to discover subsumptive relationships within sentences, as shown in Table 2. The first 5 are Hearst Patterns [16], and the last 4 are custom deviations based on observations of text in privacy policies. For each pattern, PolicyLint ensures that named-entity labels are consistent across the pattern (i.e., PolicyLint uses Hearst patterns enforcing constraints on named-entity labels). For example, Example 1 is recognized by the pattern "X, such as $Y_1, Y_2, \cdots, Y_n$" where X is a noun, $Y_1, Y_2, \cdots, Y_n$ are all nouns, and the NER labels for X and each Y_i are all data objects. Note that PolicyLint merges noun phrases before applying the lexicosyntactic patterns to ease extraction.

Given the set of extracted relationships, PolicyLint normalizes the relationships by lemmatizing the text and substituting terms with their synonym. For example, consider that "blood sugar levels" is a synonym for "blood glucose level." Lemmatization turns "blood sugar levels" into "blood sugar level," and synonym substitution turns it into "blood glucose level." To identify synonyms, we output non-terminal (i.e., X value of the Hearst patterns) data objects and entities in the subsumptive relationships. We manually scan through the list and mark synonyms. We repeat the process with the terminal nodes that are included after constructing the ontology. We then output the data objects and entities labeled from all policies and sort the terms by frequency. We mark synonyms for the most frequent terms by keyword searching for related terms based on sub-strings and domain knowledge. For example, if "location" appears as a frequent term, we output all data objects that contain the word "location," read through the list, and mark synonyms (e.g., "geographic location"). Next, we use domain knowledge to identify that "latitude and longitude" is a synonym of "location," output the terms that contain those words, and manually identify synonyms.

Table 3: Seed terms used for ontology construction

Ontology	Seeds
Data Ontology	information, personal information, non-personal information, information about you, biometric information, financial information, device sensor information, government-issue identification information, vehicle usage information
Entity Ontology	third party

2.1.3 Ontology Construction

PolicyLint generates ontologies by combining the subsumptive relationships extracted from policies with a set of seed terms (Table 3). For each ontology, PolicyLint iterates through each of the seeds, selecting relationships that contain it. PolicyLint then expands the term list from the relationships in that iteration. PolicyLint continues iterating over the relationships until no new relationships are added to the ontology. If there exists any inconsistent relationship where X is subsumed under Y and Y is subsumed under X, PolicyLint uses the relationship that has a higher frequency (i.e., appearing in more privacy policies). Once a fixed point is reached, PolicyLint ensures that there is only one root node by creating connections between any nodes that do not contain inward-edges with the root of the ontology (i.e., "information" for the data ontology, and "public" for the entity ontology). Finally, PolicyLint ensures that no cycles exist in the ontology by identifying simple cycles in the graph and removing an edge between nodes to break the cycle. PolicyLint chooses which edge to remove by finding the edge that appears least frequently in the subsumptive relationships and ensures that the destination node has more than one in-edge to ensure that a new root node is not created.

2.2 Policy Statement Extraction

The goal of policy statement extraction is to extract a concise representation of a policy statement to allow for automated reasoning over this statement. We represent data sharing and collection statements as a tuple (*actor*, *action*, *data object*, *entity*) where the *actor* performs some *action* (i.e., share, collect, not share, not collect) on the *data object*, and the *entity* represents the entity receiving the data object. For example, the statement, "We will share your personal information with advertisers," can be represented by the tuple of (we, share, personal information, advertisers). PolicyLint extracts complete policy statements from privacy policy text by using patterns of the grammatical structures between data objects, entities, and verbs that represent sharing or collection (for brevity we call these verbs SoC verbs). This section describes the steps in policy statement extraction.

2.2.1 DED Tree Construction

The goal of constructing the data and entity dependency (DED) trees is to extract a concise representation of the grammatical relationships between the data objects, entities, and SoC verbs (i.e., the verbs that represent sharing or collection). The main intuition behind constructing these trees is to allow PolicyLint to infer semantics of the sentence based on the grammatical relationships between the tokens (i.e., *who* collects/shares *what* with *whom*). The DED tree for a sentence is derived from the sentence's dependency-based parse tree. However, the DED tree removes nodes and paths that are not relevant to the data objects, entities, or SoC verbs, and performs a set of simplifications to generalize the representation. The transformation for Example 2 is shown in Figure 2.

Table 4: SoC verbs used by PolicyLint

Type	Word
Sharing	disclose, distribute, exchange, give, provide, rent, report, sell, send, share, trade, transfer, transmit
Collection	access, check, collect, gather, know, obtain, receive, save, store, use

Example 2. *If you register for our cloud-based services, we will collect your email address.*

To construct DED trees, PolicyLint parses a sentence and uses its custom-trained NER model to label data objects and entities within the sentence (Section 2.1). PolicyLint merges noun phrases and iterates over sentence tokens to label SoC verbs by ensuring that the PoS (part-of-speech) tag of the token is a verb and the lemma of the verb is in PolicyLint's manually curated list of terms (Table 4). PolicyLint also labels the pronouns, "we," "I," "you," "me," and "us," as entities during this step. PolicyLint then extracts the sentence's dependency-based parse tree whose nodes are labeled with the data object, entity, and SoC verb labels as discussed earlier.

Negated Verbs: PolicyLint identifies negated verbs by checking for negation modifiers in the dependency-based parse tree. If the verb is negated, PolicyLint labels the node as negative sentiment. PolicyLint propagates the negative sentiment to descendant verb nodes in three cases. First, if a descendant verb is part of a conjunctive verb phrase with the negated verb, negative sentiment is propagated. For example, "*We do not sell, rent, or trade your personal information,*" means "not sell," "not rent," and "not trade." Second, if the descendant verb has an open clausal complement to the negated verb, negative sentiment is propagated. For example, "*We do not require you to disclose any personal information,*" initially has "require" marked with negative sentiment. Since "disclose" is an open clausal complement to "require," it is marked with negative sentiment. Third, if the descendant verb is an adverbial clause modifier to the negated verb, negative sentiment is propagated. For example, "*We do not collect your information to share with advertisers,*" initially has "collect" marked with negative sentiment. Since "share" is an adverbial clause modifier to "collect," "share" is marked with negative sentiment.

Exception Clauses: PolicyLint identifies exception clauses by traversing the parse tree and finding terms that represent exceptions to a prior statement, such as "except," "unless,"

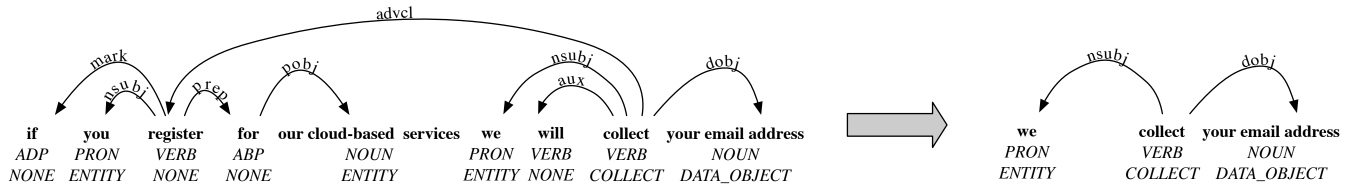


Figure 2: Transformation of Example 2 from its dependency-based parse tree to its DED tree.

"aside/apart from," "with the exception of," "besides," "without," and "not including." For each identified exception clause, PolicyLint traverses down the parse tree from the exception clause to identify verb phrases (subject-verb-object) and noun phrases related to that exception. PolicyLint then traverses upward from the exception term to identify the nearest verb node and appends as a node attribute the list of noun phrases and verb phrases identified in the downward traversal.

In certain cases, the term may not have a subtree. For example, the exception term may be a *marker* that introduces a subordinate clause. In the sentence, "*We will not share your personal information unless consent is given,*" the term "unless" is a marker that introduces the subordinate clause "your consent is given." For empty sub-trees, PolicyLint attempts the downward traversal from its parent node.

DED Tree construction: Finally, PolicyLint constructs the DED tree by computing the paths between labeled nodes on the dependency-based parse tree, copying labels and attributes described above. Note that PolicyLint also copies over all unlabeled subjects and direct objects from the parse tree, as they are needed to extract the information. PolicyLint further simplifies the tree by merging conjuncts of SoC verbs into one node if the coordinating conjunction is "and" or "or." For example, "*We will not sell, rent, or trade your personal information,*" can be simplified by collapsing "sell," "rent," and "trade" into one node. The resulting node's label is a union of all of the tags of the merged verbs (i.e., {share} + {collect} = {share, collect}. Similarly, PolicyLint repeats the same process for conjuncts of data objects and entities.

PolicyLint then prunes the DED tree by iterating through the nodes labeled as verbs in the graph and performing the following process. First, for a verb node labeled as an SoC verb, PolicyLint ensures that its sub-tree contains at least one other node labeled as an SoC verb, data object, or entity. If the node's sub-tree does not meet this condition, PolicyLint removes the subtree rooted at the node labeled as an SoC verb. Second, for verb nodes not labeled as SoC verbs, PolicyLint ensures that at least one SoC verb is contained in its sub-tree and that it meets the preceding conditions for an SoC verb. Similarly, if these conditions are not met, PolicyLint also removes the sub-tree rooted at that non-labeled verb node. For example, this pruning step causes the sub-tree rooted at the verb "register" to be removed in Figure 2.

2.2.2 SoC Sentence Identification

To identify sentences that describe sharing and collection practices, PolicyLint takes a set of positive examples of sentences as input and then extracts their DED trees to use as known patterns for sharing and collection phrases. In particular, we start by feeding PolicyLint a set of 560 example sentences that describe sharing and collect practices. PolicyLint generates the DED trees from these sentences and learns 82 unique patterns. The example sentences are auto-generated from a set of 16 sentence templates (Appendix B). We choose to auto-generate the sentences, because it is challenging to manually select a set of sentences with diverse grammatical structures. Our auto-generation does not adversely impact PolicyLint's extensibility, as adding a new pattern is as simple as feeding PolicyLint a new sentence for reflecting this new pattern.

PolicyLint iterates through each sentence of a given privacy policy. If the sentence contains at least one SoC verb and data object (labeled by NER), PolicyLint constructs the DED tree. PolicyLint then compares the sentence's DED tree to the DED tree of each known pattern. A pattern is matched if (1) the label types of the sentence's DED tree are equivalent to the ones of the known pattern's DED tree (e.g., {*entity*, *SoC_verb*, *data*}), and (2) the known pattern's DED tree is a subtree of the sentence's DED tree.

For a tree t_1 to be a subtree of tree t_2, (1) the tree structure must be equivalent, (2) the dependency labels on edges between nodes must match, and (3) the following three node conditions must hold. First, for SoC verb nodes to match, they must have a common lemma. For example, a node with the lemmas {*sell*, *rent*} matches a node with lemma {*rent*}. Second, if the node's part-of-speech is an apposition, the tags, dependency label, and lemmas must be equal. Third, for all other nodes, the tags and dependencies must be equal.

On sub-tree match, PolicyLint records the nodes in the sub-tree match and continues the process until either (1) each pattern is checked, or (2) the entire DED tree has been covered by prior sub-tree matches. If at least one sub-tree match is found, PolicyLint identifies the sentence as a potential SoC sentence and begins extracting the policy statement tuple.

2.2.3 Policy Extraction

The goal of policy extraction is to transform the DED tree into a (*actor*, *action*, *data object*, *entity*) tuple for a policy statement. PolicyLint performs extraction starting with the SoC nodes present in the sub-tree matches. If multiple SoC nodes exist in the sub-tree matches, multiple tuples are generated. However, multiple sub-tree matches over the same SoC node result in the generation of only one tuple. The SoC determines the action (e.g., collect, not_collect). The action's sentiment is determined based on whether the node is labeled with positive or negative sentiment, as discussed in Section 2.2.1.

Actor Extraction: To extract the actor, PolicyLint starts from the matching SoC verb node. The actor is a labeled entity chosen from the (1) subject, (2) prepositional object, or (3) direct object (in that order). However, if the dependency is an open clausal complement or adverbial clause modifier, PolicyLint prioritizes the direct object and prepositional object over the subject. For example, "*We do not require you to disclose any personal information,*" has "disclose" as an open clausal complement to "require." In this case, the correct actor of this policy statement is the user (i.e., "you") rather than the vendor (i.e., "we"), which is correctly captured due to PolicyLint's dependency-based prioritization rules. If no match is found, PolicyLint traverses up one level in the DED tree and repeats. Finally, if no match is found, PolicyLint assumes that the actor is the implicit first party.

Data Object Extraction: To extract the data objects, PolicyLint starts from the matching SoC verb node. It traverses down the DED tree to extract all nodes labeled as data objects. The traversal continues until another SoC verb is reached. If no data objects are found, and the verb's subject and direct object are not labeled as a data object, PolicyLint extracts the data objects from the nearest ancestor SoC verb.

Entity Extraction: To extract the entities, PolicyLint starts from the matching SoC verb node. It traverses down the DED tree extracting all nodes labeled as entities that are not actors. The traversal continues until another SoC verb is reached.

Exception Clauses: PolicyLint considers exception clauses if the verb is marked with negative sentiment (e.g., not collect, not share), creating a cloned policy statement with the sentiment to change. We do not handle exception clauses for positive sentiment. For example, "*We might also share personal information without your consent to carry out your own requests,*" still shares personal information.

For negative sentiment verbs, there are three cases. First, if the exception clause's node attribute contains only data objects, PolicyLint replaces the data objects of the new policy with the data objects under the exception clause. For example, "*We will not collect your personal information except for your name and phone number,*" produces policies: (we, not_collect, personal information, NULL), (we, collect, [name, phone number], NULL). Second, if all noun phrases have an entity label, PolicyLint replaces the entities of the new policies with the entities under the exception attribute. For example, "*We do not share your demographics with advertisers except for AdMob,*" produces policies: (we, not_share, demographics, advertisers) and (we, share, demographics, AdMob). Third, if the labels are not data objects or entities, PolicyLint removes the initial policy statement. For example, "*We will not collect your personal information without your consent,*" produces the policy: (we, collect, personal information, NULL).

Policy Expansion: PolicyLint may extract multiple actors, actions, data objects, and entities when creating policy statements. These complex tuples are expanded. For example, ([we], [share, sell], [location, age], [Google, Facebook]) is expanded to (we, share, location, Google), (we, share, location, Facebook), (we, share, age, Google), etc.

2.3 Policy Contradictions

PolicyLint's components of ontology generation and policy extraction identify the sharing and collection statements in privacy policies. This section formally defines a logic for characterizing different contradictions. It then describes how PolicyLint uses this logic to identify candidate contradictions within privacy policies. We note that contradictions may occur between an app's privacy policy and the privacy policies of third-party libraries (e.g., advertisement libraries). While our study focuses specifically on contradictions within an individual privacy policy, the formal logic and subsequent analysis tools may also be used to include the privacy policies for third-party libraries with minimal modification.

2.3.1 Policy Simplification

PolicyLint simplifies policy statements for contradiction analysis. We refer to the (*actor*, *action*, *data object*, *entity*) tuple defined in Section 2.2 as a Complete Policy Statement (CPS). We simplify CPS statements about the sharing of data (i.e., *action* is share or not share) by capturing sharing as collection.

Definition 1 (Simplified Policy Statement: SPS)**.** *An SPS is a tuple, $p = (e, c, d)$, where d is the data object discussed by the statement, $c \in \{collect, not\_collect\}$ represents whether the object is collected or not collected, and e is the entity receiving the data object.*

To transform a CPS into an SPS, we leverage three main insights. First, policies do not typically disclose whether the sharing of the data occurs at the client side or server side. Therefore, an actor sharing a data object with an entity may imply that the actor is collecting the data and performing the data sharing at the server side. In this case, a new policy statement would need to be generated for allowing the actor to collect the data object (Rule T1, Table 5). Second, a data object being shared with an entity may imply that the entity is collecting the information from the mobile device (Rule T2, Table 5). Similarly, a policy for stating that the actor does not share a data object with an entity implies that the entity is not collecting the data from the mobile device (Rule T3, Table 5). Finally, a policy for stating that the actor does not share a data object implies that the actor collects the data object, because

Table 5: Rules that transform a CPS into an SPS

Rule	Transformation Rules	Rationale
T1	(actor, share, data object, entity) $\implies$ (actor, collect, data object)	Unknown whether sharing occurs at the client side or server side
T2	(actor, share, data object, entity) $\implies$ (entity, collect, data object)	Can observe only client-side behaviors
T3	(actor, not_share, data object, entity) $\implies$ (entity, not_collect, data object)	Can observe only client-side behaviors
T4	(actor, not_share, data object, entity) $\implies$ (actor, collect, data object)	If mention not share, assume implicit collection

the policy would likely have not mentioned not sharing data that was never collected (Rule T4, Table 5).

However, there are two special cases. First, PolicyLint treats only verb lemmas "save" and "store" with positive sentiment ("not saving/storing" does not mean "not collecting"). Second, PolicyLint ignores negative statements with verb lemma "use." This case leads to false positives, as PolicyLint does not extract the collection purpose. For example, "*We do not use your location for advertising*," means that it is not collected for the specific purpose of advertising.

2.3.2 Contradiction Types

We model an app's privacy policy as a set of simplified policy statements P. Let D represent the total set of data objects and E represent the total set of entities, as represented by ontologies for data objects and entities, respectively. A policy statement $p \in P$ is a tuple, $p = (e, c, d)$ where $d \in D$, $e \in E$, and $c \in \{collect, not\_collect\}$ (Definition 1).

Language describing policy statements may use different semantic granularities. One policy statement may speak in generalizations over data objects and entities while another statement may discuss specific types. For example, consider the policies $p_1 = $ (advertiser, not_collect, demographics) and $p_2 = $ (Google Admob, collect, age). If we want to identify contradictions, we need to know that Google AdMob is an advertiser and age is demographic information. 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 age is subsumed under demographics).

We use the following notation to describe binary relationships between terms representing data objects and entities.

Definition 2 (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 (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 such that $x \not\equiv_o y$. Similarly, $x \sqsubseteq_o y \implies x \sqsubset_o y \vee x \equiv_o y$.*

Note that Definitions 2-3 parameterize the operators with an ontology o. PolicyLint operates on two ontologies: data objects and entities. Therefore, the following discussion parameterizes the operators with δ for the data object ontology and ε for the entity ontology. For example, $x \equiv_\delta y$ and $x \equiv_\varepsilon y$.

A contradiction occurs if two policy statements suggest that entities both may and may not collect or share a data object.

Table 6: Contradictions (C) and Narrowing Definitions (N) $P = \{(e_i, \text{collect}, d_k), (e_j, \text{not\_collect}, d_l)\}$

Rule	Logic	Example
C_1	$e_i \equiv_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(companyX, collect, email address) (companyX, not_collect, email address)
C_2	$e_i \equiv_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(companyX, collect, email address) (companyX, not_collect, personal info)
C_3	$e_i \sqsubset_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(companyX, collect, email address) (advertiser, not_collect, email address)
C_4	$e_i \sqsubset_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(companyX, collect, email address) (advertiser, not_collect, personal info)
C_5	$e_i \sqsupset_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(advertiser, collect, email address) (companyX, not_collect, personal info)
N_1	$e_i \equiv_\varepsilon e_j \wedge d_k \sqsupset_\delta d_l$	(companyX, collect, personal info) (companyX, not_collect, email address)
N_2	$e_i \sqsubset_\varepsilon e_j \wedge d_k \sqsupset_\delta d_l$	(companyX, collect, personal info) (advertiser, not_collect, email address)
N_3	$e_i \sqsupset_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(advertiser, collect, email address) (companyX, not_collect, email address)
N_4	$e_i \sqsupset_\varepsilon e_j \wedge d_k \sqsupset_\delta d_l$	(advertiser, collect, personal info) (companyX, not_collect, email address)

Contradictions can occur at the same or different semantic levels. For example, the simplest form of contradiction is an exact contradiction where a policy states that an entity will both collect and not collect the same data object, e.g., (advertiser, collect, age) and (advertiser, not_collect, age). Due to subsumptive relationships (Definitions 3), there are 3 relationships between terms ($x \equiv_o y$, $x \sqsubset_o y$, and $x \sqsupset_o y$). Each binary relationship applies to both entities and data objects. Thus, there are $3^2 = 9$ types of contradictions, as shown in Table 6.

Contradictions have two primary impacts when privacy policies are analyzed. First, all contradictions impact analysis techniques that seek to automatically reason over policies and may result in these techniques making incorrect or inconsistent decisions. For example, unlike firewall rules that have a specific evaluation sequence, privacy policy statements do not have a specific pre-defined sequence of evaluation. Therefore, analysis techniques may make incorrect or inconsistent decisions based on the order in which they evaluate policy statements. Second, contradictions may impact a human analyst's understanding of a privacy policy, such as by containing misleading statements. We define two groupings of contradictions based on whether they may impact a human's comprehension of privacy policies or solely impact automated analysis.

Logical Contradictions (C_{1-5}): Logical contradictions are contradictory policy statements that are more likely to cause harm if users and analysts are not aware of the contradictory statements. Logical contradictions may cause difficulties when humans attempt to comprehend or interpret the sharing

Table 7: First Party Synonyms

First Party Synonyms
we, I, us, me, our app, our mobile application, our mobile app, our application, our service, our website, our web site, our site
app, mobile application, mobile app, application, service, company, business, web site, website, site

and collection practices discussed in the policy. They can be characterized as either exact contradictions (C_1) or those that discuss *not* collecting broad types of data and later discuss collecting exact or more specific types (C_{2-5}). One example is a policy that initially claims not to collect personal information, but later in fine print discloses collecting a user's name and email address for advertisers. Similar to the goal of software `lint` tools, PolicyLint flags logical contradictions as problems within policies to allow a human analyst to manually inspect the statements and analyze intent. As shown in Section 3, these contradictions may lead to the identification of intentionally deceptive policy statements or those that may result in ambiguous interpretations.

Narrowing Definitions ($\mathbf{N}_{1-4}$): Narrowing definitions are contradictory policy statements where broad information is stated to be collected, and specific data types are stated not to be collected. These statements narrow the scope of the types of data that are collected, such as stating that personal information is collected while your name is *not* collected. Note that narrowing definitions are not necessarily an undesirable property of policies, because saying that a broad data type is collected does not necessarily imply that every specific subtype is collected, but narrowing definitions may result in vague policies. There may be clearer ways for policy writers to convey this information, such as explicitly stating the exact data types collected and shared. For example, if the app collects your email address, the policy could directly state, "We collect your email address," instead of including a narrowing definition, such as "We collect personal information. We do not collect your name." However, policies that narrow the scope of their data sharing and collection practices can be seen as more desirable in contrast to policies that just disclose practices over broad categories of data. Nonetheless, narrowing definitions impact the logic behind analysis techniques, as they must consider prioritization of data objects and entities.

2.3.3 Contradiction Identification

PolicyLint uses the contradiction types from Table 6 to determine a set of candidate contradictions. It then uses a set of heuristics to reduce the set of candidate contradictions that are potentially low-quality indicators of underlying problems. Next, PolicyLint prepares the contradictions for presentation by collapsing duplicate contradictions, linking other metadata (e.g., download counts of apps), and by using a set of filtering heuristics to allow the regulator or privacy analysts to focus on specific subclasses of candidate contradictions. The remainder of this section describes this process.

Initial Candidate Set Selection: Given policy statements p_1 and p_2, PolicyLint ensures that $p_1.c$ does not equal $p_2.c$, as contradictions require opposing sentiments. PolicyLint then compares entities $p_1.e$ and $p_2.e$, determining whether they are equal or have a subsumptive relationship. A subsumptive relationship occurs if there is a path between the entities in the entity ontology. When comparing entities, PolicyLint treats the terms in Table 7 as synonyms for the first party (i.e., "we"). If an entity match is found, PolicyLint then performs the same steps for data objects $p_1.d$ and $p_2.d$ using the data object ontology. If a data object match is found, PolicyLint adds the candidate contradiction to the candidate set. Note that in policy statements PolicyLint ignores entities and data objects that are not contained in the ontologies, as it cannot reason about those relationships. However, if PolicyLint cannot find a direct match for a term in the ontologies, it will try to find submatches by splitting the term on the coordinating conjunction terms (e.g., "and," "or") and checking for their existence in the ontologies. Furthermore, PolicyLint does not identify policy statements as contradictions if they are generated from the same sentence due to the semantics of exception clauses. For example, "*We do not collect your personal information except for your name*," produces simplified policy statements (we, not_collect, personal information) and (we, collect, name). While the semantics of this statement is clear, our definition of contradictions would incorrectly identify these statements as a C_2 contradiction. Therefore, PolicyLint ignores same-sentence contradictions to reduce false positives.

Candidate Set Reduction: PolicyLint uses heuristics to prune candidate contradictions that are likely low-quality indicators of underlying problems. PolicyLint removes contradictions that occur based on potentially poor relationships discovered in the ontologies. For example, PolicyLint filters out contradictions that occur between certain data object pairs, such as "usage information" and "personal information." Contradictions whose entities refer to the user (e.g., "user," "customer," "child") or involve terms for general data objects (e.g., "information," "content") or entities (e.g., "individual," "public") are also removed. Finally, PolicyLint removes candidate contradictions where a negative-sentiment policy statement may be conditioned with age restrictions or based on user choice by searching for common phrases in the sentences that are used to generate the policy statements (e.g., "under the age of," "from children," "you do not need to provide"). Note that some of these reductions may occur during candidate set construction to reduce complexity of the analysis.

Candidate Set Filtering: PolicyLint further filters the set of candidate contradictions into subsets based on the data objects involved in the contradictions to allow for targeted exploration during verification. For example, all of the contradictions with statements involving collecting email address but not collecting personal information are placed into one subset (e.g., (*, collect, email address) and (*, not_collect, PII)).

Contradiction Validation: Given the filtered subsets of candidate contradictions, the next step is to explore certain subsets and validate candidate contradictions. To validate a candidate contradiction, the analyst reads through the policy statements and sentences that are used to generate them in context of the entire policy, and makes a decision.

3 Privacy Study

Our primary motivation for creating PolicyLint is to analyze contradictory policy statements within privacy policies. In this section, we use PolicyLint to perform a large-scale study on 11,430 privacy policies from top Android apps.

Dataset Collection: To select our dataset, we scrape Google Play for the privacy policy links of the top 500 free apps for each of Google Play's 35 app categories in September 2017. Note that the "top free apps" are based on the ranking provided by Google Play ("topselling_free" collection per app category). We download the HTML privacy policies using the Selenium WebDriver in a headless Google Chrome browser to allow for the execution of dynamic content (e.g., JavaScript). We exclude apps that do not have a privacy policy link on Google Play, those whose pages are unreachable at the time of collection, and privacy policies where the majority of the document is not English, as discussed in Appendix A. We convert the HTML policies to plaintext documents. Our final dataset consists of 11,430 privacy policies.

3.1 General Policy Characteristics

PolicyLint extract sharing and collection policy statements from 91% of the policies in our dataset (10,397/11,430). From those policies, PolicyLint extract 438,667 policy statements from 177,169 sentences that PolicyLint identifies as a sharing or collection sentence. Of those policy statements, 32,876 have negative sentiment and 405,789 have positive sentiment. In particular, 60.5% (6,912/11,430) of the policies have at least one negative-sentiment policy statement and 89.6% (10,239/11,430) of the policies have at least one positive sentiment policy statement. We explore why PolicyLint does not extract policy statements for 9% of the policies by analyzing a random subset of 100 policies and find that it is mainly due to dataset collection or preprocessing errors (Appendix C).

Finding 1: *Policies frequently contain negative sentiment policy statements that discuss broad categories of data.* For 60.5% of the policies with at least one negative sentiment policy statement, the data object "personal information" appears in 67.7% of those policies (4,681/6,912). This result demonstrates the importance of handling negative policy statements, as around 41.0% of the policies contain a negative sentiment policy statement for claiming that a broad type of data (i.e., "personal information") is not collected. Further, we measure the distance from the negation (i.e., "not") to the verb that the negation modifies, and find that 28.2% (3,234/11,430) of the policies have a distance greater than one word away. This

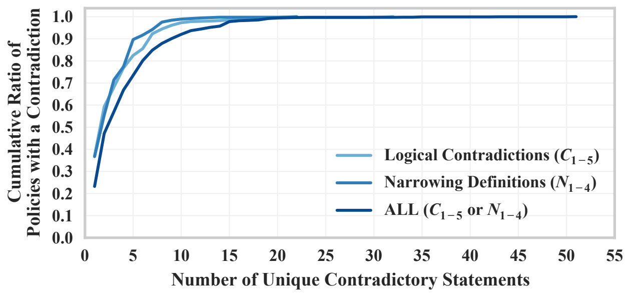


Figure 3: CDF of Contradictory Policy Statements: 50% of contradictory policies have 2 or fewer logical contradictions.

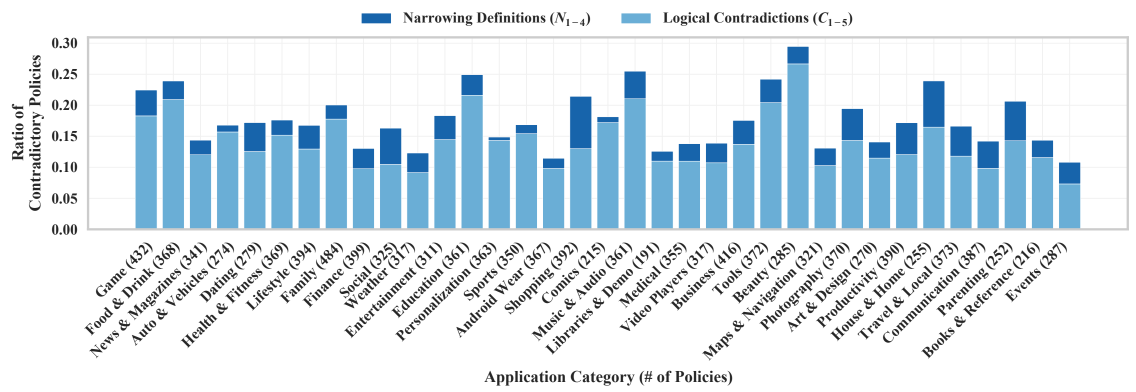
result calls into question prior approaches [26,28] that assume only positive sentiment when considering sharing and collection statements, as the prior approaches could be incorrect up to 60.5% of the time when reasoning over sharing and collection statements. Further, approaches [32] that handle negations using bigrams would have failed to reason about 28.2% of the policies.

3.2 Candidate Contradictions

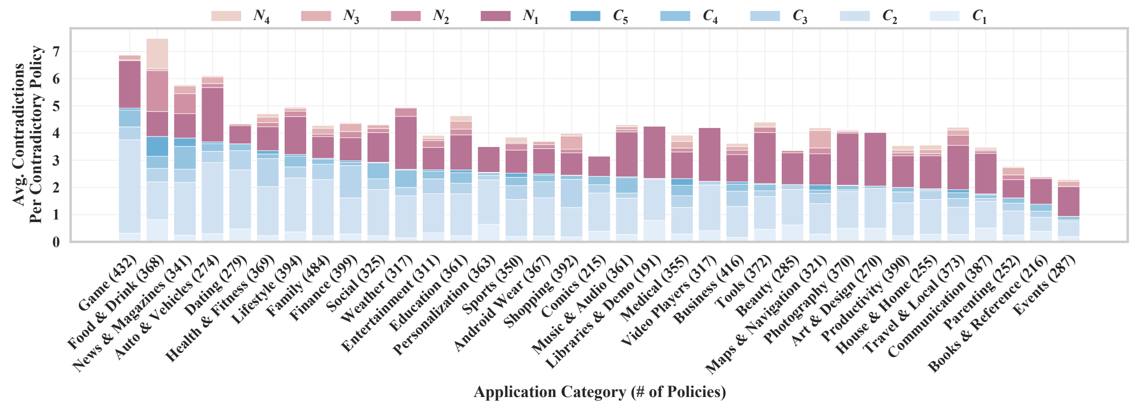
Based on PolicyLint's fine-grained policy statement extraction, we find that 59.1% (6,754/11,430) of the policies are candidates for contradiction analysis, as they contain at least one positive and one negative sentiment policy statements. Among these policies, there are 13,871 and 129,575 policy statements with negative and positive sentiment, respectively.

Finding 2: *For candidate contradictions, 14.2% of the privacy policies contain logical contradictions (C_{1-5}).* PolicyLint identifies 9,906 logical contradictions across around 14.2% (1,618/11,430) of the policies. Therefore, 14.2% of the policies may contain potentially misleading statements. Figure 3 shows that around three-fifths (59.2%) of the policies with at least one logical contradiction have 2 or fewer unique contradictions. The relatively low number of candidate contradictions per policy indicates that manual validation is feasible. As roughly 6 in 7 policies are not contradictory, writing policies without logical contradictions is possible.

Finding 3: *Contradiction prevalence and frequency do not substantially vary across Google Play app categories.* Figure 4a shows the ratio of policies containing candidate contradictions per each Google Play category. The categories with policies most and least prone to contradiction are *Beauty* and *Events*, respectively. However, when analyzing the policies within those categories, we find that their means are skewed by contradictory policies for apps by the same developer. When we recompute the means without the outliers, these categories follow the general trend. Policies with logical contradictions accompany 7.3% to 20.9% of apps across all categories. We find that policies with logical contradictions are not substantially more prevalent in particular categories of apps, but instead occur consistently in apps from every category. We also find that prevalence of logical contradiction



(a) Ratio of Contradictory Policies per Category: 79.7% of contradictory policies have at least one or more logical contradictions (C_{1-5}) that may indicate potentially deceptive statements.



(b) Average number of unique candidate contradictions per category: Logical contradictions (C_{1-5}) and narrowing definitions (N_{1-4}) are both widely prevalent across Google Play categories.

Figure 4: Distribution of Candidate Contradictions across Google Play Categories.

does not substantially vary by download count as well.

Figure 4b displays the average number of candidate contradictions for policies containing one or more contradictions. We find that frequency of logical contradiction for contradictory policies does not substantially vary across Google Play categories. Initial analysis indicates that contradictory policies for apps in the *Games* category contain around 4.9 logical contradictions on average. Further analysis reveals that this result is due to policies with 19 unique logical contradictions in 9 apps produced by the same developer, and one app that has 31 unique logical contradictions. Excluding these outliers brings the category's average to 3.16 logical contradictions per app, fitting the trend of the rest of the categories. This result may indicate that poor policies are linked to problematic developers. Similar analysis on categories *Food & Drink*, *Auto & Vehicles*, and *News & Magazines* produces similar results. We find that the number of logical contradictions per policy is roughly equivalent across app categories, indicating that one app category is not necessarily more contradictory on average than another.

Finding 4: *Negative sentiment policy statements that discuss broad categories of data are problematic.* Figure 5 shows the frequency of the most common data-type pairs referred to in contradictory policy statements. The contradictory policy statements in the topmost row are most problematic. This row represents logical contradictions, which are either (1) exact contradictions or (2) discussion of not collecting broad types of data and collecting more specific data types (C_{1-5}). As we demonstrate in Section 3.3, logical contradictions can lead to a myriad of problems when one interprets the policy including making interpretation ambiguous in certain cases. The leftmost column corresponds to narrowing definitions (N_{1-4}), which solely impact automated analysis techniques, as discussed in Section 2.3.

Finding 5: *For candidate contradictions, 17.7% of the privacy policies contain at least one or more logical contradictions (C_{1-5}) or narrowing definitions (N_{1-4}).* PolicyLint identifies 17,986 logical contradictions and narrowing definitions across around 17.7% (2,028/11,430) of the policies. Figure 3 shows that slightly more than half (57.0%) of the contradictory policies have 3 or fewer unique logical contradictions and narrowing definitions. As discussed in Section 2.3, logical contradictions and narrowing definitions impact approaches that seek to automatically reason over privacy policies. To

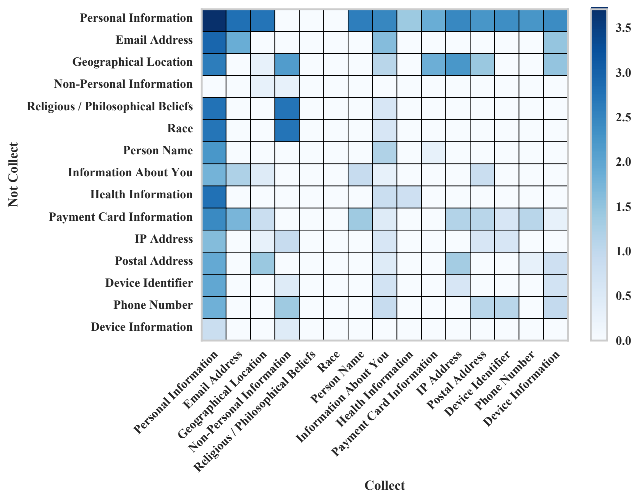


Figure 5: Log 10 Frequency of Data-Type Pairs in Contradictions: Negative statements that discuss broad categories of data are problematic (i.e., C_{1-5}).

correctly reason over contradictions, analysis techniques must include logic to prioritize specificity of data types and entities, and be able to identify potentially undecidable cases, such as exact contradictions (C_1). No prior approaches [26, 28, 29, 32] that attempt to reason over policies operate at the required granularity to identify contradictions or contain the logic to correctly reason over them. Therefore, these prior approaches could make incorrect or inconsistent decisions around 17.7% of the time around 1–3 times per policy on average. We perform similar analysis across categories as Finding 3 and find that logical contradictions and narrowing definitions do not substantially vary across Google Play categories.

3.3 Deeper Findings

In this section, we describe the findings from validating candidate contradictions. We limit our scope to logical contradictions (C_{1-5}), as they may be indicative of misleading policy statements. Due to resource constraints, we do not validate all 9,906 candidate logical contradictions from the 1,618 policies. Instead, we narrow the scope of our study by choosing categories of candidate contradictions to focus on. Our selection and validation methodology is described below.

Selection Methodology: To select the categories of candidate contradictions to focus on, we analyze Figure 5 for the data objects involved in the contradictory statements. We limit our scope to logical contradictions (C_{1-5}) that discuss not collecting "personal information" and collecting "email address," "device identifier," or "personal information." We also explore two other categories of candidate contradictions in which one type of data can be derived from the other type; these categories caught our attention when analyzing the heat map. Within each category of candidate contradictions, we choose which candidate contradictions to validate by sorting the contradictions based on the belonging app's popularity and working down the list. We spend around one week validating contradictions where our cutoffs are due to time constraints and attempting to achieve coverage across categories.

Validation Methodology: To validate candidate contradiction, one of three student authors reads through the sentences that are used to generate each policy statement for the candidate contradiction to ensure correctness of policy statement extraction. If there is an error with policy statement extraction, we record the candidate contradiction as a false positive and stop analysis. Next, we locate the sentences within the policy and view the context in which they appear (i.e., section, surrounding sentences) to determine whether the policy statements are contradictory. We try to determine why the contradiction occurs if possible and record any observations. If the author is uncertain about his/her decision, a second author analyzes it. The two authors discuss and resolve conflicts, with no conflicts left unresolved after discussion.

3.3.1 Personal Information and Email Addresses

For candidate contradictions with negative statements about "personal information" and with positive statements about "email address," we find 618 candidate contradictions across 333 policies (C_2, C_4, C_5) We validate 204 candidate contradictions from 120 policies. We find that 5 candidate contradictions are false positives due to inaccuracies of labeling data objects by the NER model. From the 199 remaining candidate contradictions across 118 policies, we have the following main findings. Note that for the findings discussed below, the terms "personally identifiable information" and "personal information" are commonly used synonymously in USA regulations, and "personal data" is considered the EU equivalent albeit covering a broader range of information.

Finding 6: *Policies are stating certain types of common personally identifiable information, such as email addresses, as non-personally identifiable.* When validating 14 candidate contradictions, we find 14 policies for explicitly stating that they do NOT consider email address as personally identifiable information. 11 of those policies are released by the same developer (OmniDroid) where the most popular app in the set (com.omniluxtrade.allrecipes) has over 1M+ downloads. OmniDroid's policy explicitly lists email address when defining non-personally identifiable information. The remaining 3 policies belong to another app developer, PlayToddlers. The apps are explicitly targeted toward children from 2-8 years old and have between 500K-1M+ downloads for each app. Their policy states the following sentence verbatim, "When the user provides us with an email address to subscribe to the "PlayNews" mailing list, the user confirms that this address is not a personal data, nor does it contain any personal data."

The fact that *any* privacy policies are declaring email addresses as non-personal information is surprising, as it goes against the norms of *what* data is considered personal information as defined by regulations (e.g., CalOPPA, GDPR), standards bureaus (NIST), and common sense.

Finding 7: *Services that auto-generate template-based policies for app developers are producing contradictory policies.* During our validation process, we notice that many policies have similar structural compositions and contain a lot of the same text in paragraphs. When validating 78 candidate contradictions, we find 59 contradictory policies that are automatically generated or used templates. Identical policy statements from various developers suggest that some policies may be generated automatically or acquired from a template. We investigate these cases and identify 59 policies that use 3 unique templates. We check that these policies are not ones for apps created by the developers or organization. Findings 8 and 11 discuss the problems caused by the templates. This result demonstrates that poor policy generators can be a contributing factor for numerous contradictory policies.

Finding 8: *Policies use blanket statements affirming that personal information is not collected and contradict themselves by stating that subtypes of personal information are collected, such as email addresses.* When validating 182 candidate contradictions, we find 104 policies for broadly making blanket statements that personal information is not collected in one part of the policy and then directly contradicting their prior statements by disclosing that they collect email addresses. We find 69 of those policies (127 validated contradictions) for stating that they do not collect personal information, but later stating that they collect email addresses for some purpose. Of those 69 policies, 32 policies define email address as personal information in one part of their policy. Due to the lack of definition of what they consider personal information in the other 37 policies, it is unclear whether they do not consider email address as personal information or are just contradictory.

The same organization (emoji-keyboard.com) produces 20 of those policies that explicitly define email addresses as personal information, but contradict themselves. The most popular app in that group has 50M+ downloads (emoji.keyboard.emoticonkeyboard). The following two sentences are in the policy verbatim: (1) *"Since we do not collect Personal Information, we may not use your personal information in any way."*; (2) *"For users that opt in to Emoji Keyboard Cloud, we will collect your email address, basic demographic information and information concerning the words and phrases that you use ("Language Modeling Data") to enable services such as personalization, prediction synchronization and backup."* This case is clearly a contradictory statement and arguably a misleading practice.

A policy for a particular app with 1M+ downloads (com.picediting.haircolorchanger) appears to have been potentially trying to mislead users by using bold text to highlight desirable properties and then contradicting themselves. For example, the following excerpt is in the policy verbatim including the bold typography: *"**We do not collect any Personal information** but it may be collected in a number of ways. We may collect certain information that you voluntarily provide to us which may contain personal information. For example, we may collect your name, email address you provide us when you contact us by e-mail or use our services..."* The use of bold typography and general presentation of these policy statements could potentially be considered as attempting to deceive the reader, who may not perform a close read of the text in fine-print. This finding validates PolicyLint's value in flagging problematic areas in policies to aid in the identification of deceptive statements.

Finding 9: *Policies consider hashed email addresses as pseudonymized non-personal information and share it with advertisers.* When validating three candidate contradictions, we find that two policies discuss sharing hashed email addresses with third parties, such as advertisers. One candidate contradiction is a false positive due to misclassifying a sentence discussing opt-out choices as a sharing or collection sentence. The other policy belongs to an app named Tango (com.sgiggle.production). Tango is a messaging and video call app, which has over 100M+ downloads on Google Play and according to their website has 390M+ users globally. Their policy states the following sentences verbatim ,*"For example, we may tell our advertisers the number of users our app receives or share anonymous identifiers (such as device advertising identifiers or hashed email addresses) with advertisers and business partners."* Tango explicitly states that they consider hashed email addresses as anonymous identifiers. It is arguable whether hashing is sufficient for pseudonymization as defined by GDPR, as it is likely that advertisers are using hashed email addresses to identify individuals.

3.3.2 Personal Information and Device Identifiers

For the candidate contradictions with negative statements about "personal information" and with positive statements about "device identifiers," we find 234 candidate contradictions across 155 policies. We investigate this group of candidate contradictions as there are differing regulations across countries on whether device identifiers are considered personal information. For example, various court cases within the US (Robinson v. Disney Online, Ellis v. Cartoon Network, Eichenberger v. ESPN) rule that device identifiers are not personal information. However, the GDPR defines device identifiers as personal information. Therefore, our goal is to check whether policies are complying to the stricter GDPR definition of personal information or to the US definition, as the outcome could hint toward problems with complying to regulations across country boundaries. In total, we validate 10 candidate contradictions across 9 policies.

Finding 10: *Policies are considering device identifiers as non-personal information, raising concerns regarding globalization of their policies.* When validating 10 candidate contradictions, we find 9 policies for stating that they do not collect personal information, but later state that they collect device identifiers. We find that classification of device identifiers varies across policies. We find 4 policies that explicitly describe device identifiers as non-personal information. The most popular app is Tango (com.sgiggle.production), which boasts of 390M+ global users on their website. It is likely a safe assumption that some of those users are in the EU, which is subject to GDPR. As their current policy still contains this statement, it may hint that they may not be GDPR compliant.

To reduce the threats to the validity of our claims, we re-request the 9 policies using a proxy to route the traffic through an EU country (Germany) to ensure that an EU-specific policy is not served based on the origin of the request. We request the English version of the policy where applicable and find similarly problematic statements in regard to not treating device identifiers as personal information.

3.3.3 Personal Information

We find 5100 candidate contradictions across 1061 policies where the data type of both the negative statement and positive statement is "personal information." We validate 254 candidate contradictions across 153 policies.

Finding 11: *Policies directly contradict themselves.* When validating the 254 candidate contradictions, we find that the 153 policies directly contradict themselves on their data practices on "personal information." For example, the policy for an app with 1M+ downloads states "We may collect personal information from our users in order to provide you with a personalized, useful and efficient experience." However, later in the policy they state, "We do not collect Personal Information, and we employ administrative, physical and electronic measures designed to protect your Non-Personal Information from unauthorized access and use." These scenarios are clearly problematic, as the policies state both cases and it makes it difficult, if not, impossible to determine their actual data sharing and collection practices.

3.3.4 Derived Data

In this section, we explore cases of candidate contradictions where the negative statements discuss data that can be derived from the data discussed in the positive statement. In particular, we explore two cases: (1) coarse location from IP address; and (2) postal address from precise location.

For "coarse location from IP," we find 170 candidate contradictions from 167 policies that represent collecting IP address and not collecting location. We remove candidate contradictions whose statements discuss precise location, as IP address does not provide a precise location. This filtering results in 18 candidate contradictions from 18 different policies. We validate 15 candidate contradictions across 15 different policies for this case. We note that 3 candidate contradictions from 3 policies are false positives due to incorrect negation handling.

For "postal address from precise location," we find 27 candidate contradictions across 20 policies. Note that we remove candidate contradictions that discuss coarse location, as they are not precise enough to derive postal addresses. We validate 22 candidate contradictions across 17 apps, as 5 candidate contradictions are false positives due to sentence misclassification (4 cases) or errors of handling negations (1 case).

Finding 12: *Policies state that they do not collect certain data types, but state that they collect other data types in which the original can be derived.* When validating the 15 candidate contradictions for "coarse location from IP," we find that all 15 policies are stating that they do not collect location information, but state that they automatically collect IP addresses. As coarse location information can generally be derived from the user's IP address, it can be argued that the organization is technically collecting the user's location information. Interestingly, two of the policies discuss that if users disable location services, then location will not be collected. It is highly unlikely that companies cease IP address collection based on device privacy settings. However, as IP address collection typically occurs passively at the server side, we cannot claim with 100% certainty that the companies still collect IP addresses when location services are disabled.

When validating 20 candidate contradictions for "postal address from precise location," we find that 15 policies discuss not collecting postal addresses, but then state that they collect locations. Similar to the preceding case, postal addresses can be derived from location data (i.e., latitude and longitude). Again, the argument can be made that they are collecting data precise enough to be considered a postal address, causing a contradiction. For the other two candidate contradictions from two policies, it is not clear whether it is actually a contradiction, as they state that they do not collect addresses from the user's address book, which is more specific than a general statement about not collecting addresses.

3.4 Notification to Vendors

For the 510 contradictions that are validated across 260 policies, we contact each vendor via the email address listed for the privacy policy's corresponding app on Google Play. We disclose the exact statements that we find to be contradictory and explain our rationale. We ask whether they consider the statements to be contradictory and request clarifications on their policy. Figure 6 (Appendix) shows a template of the email. Overall, 244 emails are successfully delivered, as 16 email addresses are either invalid or unreachable. In total, we receive a 4.5% response rate (11/244), which is relatively substantial when considering that responding to our emails could raise liability concerns. All of the responses are received within a week or less after we send the initial emails. We have not received additional responses in the 4 months that have passed before publication. The remainder of this section discusses the responses.

Fixed Policy: Three vendors agree with our findings and update their policy to remove the contradiction. One vendor states that there is an "error" in their privacy policy and updates it accordingly. We confirm that the policy has been updated. The remaining two vendors state that the self-contradictory portion of the policy is a leftover remnant from a prior update and should have been removed. They clarify that they do not collect email addresses.

Disagreed with our Findings: One vendor explicitly disagrees with our findings. Their policy states that they do not collect personal information, but also states that they collect email addresses. The vendor responds by claiming that email addresses are only personal information if it contains identifiable information, such as your name (e.g., john.doe@gmail.com), but are not personal information if it does not contain identifiable information (e.g., wxyz@gmail.com). They state that they explicitly tell users not to submit email addresses that contain personal information and thus their policy is not contradictory. This interpretation is surprising, because it goes against the definition of email addresses as personal information, as defined by regulations (e.g., CalOPPA, GDPR) and standards bureaus (NIST).

Claimed Outdated Policy: Four vendors respond that they do not find the reported statements in their current policy and that we analyzed an older version of their policy. One of the vendors sent us their updated policy for their mobile app. However, the policy has a link to refer to their full privacy policy, which links to the policy analyzed by us. We request clarifications on their "full policy," but receive no response. We analyze the remaining three policies and find that they have the contradictory statements removed from their policy.

No Comment: Two vendors respond without providing a comment or clarification of their policy. One developer simply replies back "Thanks for the observation." The other responds that their app is removed from Google Play.

4 Limitations

PolicyLint provides a set of techniques to extract a concise structured representation of both collection and sharing statements from the unstructured natural language text in privacy policies. In so doing, it provides unprecedented analysis depth, and our findings in Section 3 demonstrate the utility and value of such analysis. However, extracting structured information from unstructured natural language text continues to be an open and active area of NLP research, and there are currently no perfect techniques for this challenging problem. PolicyLint is thus limited by the current state of NLP techniques, such as the limitations of NLP parsers and named-entity recognition. Its performance also depends on its verb lists and policy statement patterns, which may be incomplete despite our best efforts, reducing overall recall. We note that as a `lint` tool, our goal is to provide high precision at a potential cost to high recall. We note that PolicyLint achieves 97.3% precision (496/510) based on the 14 false positives identified during our validation of contradictions.

Another limitation is that PolicyLint cannot extract the conditions or purposes behind collection and sharing statements. Extracting such information would allow for a more holistic analysis, but doing so would require advances in decades-old NLP problems, including semantic role labeling, coreference resolution, and natural language understanding.

Finally, our analysis focuses on policies of Android apps. While we cannot claim that our findings would certainly generalize to policies from other domains (e.g., iOS, web), we hypothesize that self-contradictions likely occur across the board, as policies are written for all platforms in largely the same way to describe data types collected by all platforms.

5 Related Work

Recent research has increasingly focused on automated analysis of privacy policies. Various approaches have used NLP for deriving answers to a limited number of binary questions [31] from privacy policies, applied topic modeling to reduce ambiguity in privacy policies [27], and used data mining [30] or deep learning [15] models to extract summaries from policies of what and how information is used. Some other approaches [26, 28] have used *crowdsourced* ontologies for policy analysis. These approaches are often limited by lack of accuracy, completeness, and collection complexity. Other approaches [12, 18] identify patterns to extract subsumptive relationships. However, they do not provide methodology to generate usable ontologies from such extracted information and rely on a fixed lexicon. Prior research has also attempted to infer negative statements in privacy policies with limited success. Zimmeck et al. [32] and Yu et al. [29] rely on keyword-based approaches of using bi-grams and verb modifiers, respectively, to detect the negative statements. In contrast to these previous approaches, our work provides a more comprehensive analysis with an automatically constructed ontology and accounting for negations and exceptions in text.

Prior approaches [2, 5, 29] identify the possibility of conflicting policy statements. However, we are the first to characterize and automatically analyze self-contradictions resulting from interactions of varying semantic levels and sentiments.

Analyzing the usability and effectiveness of privacy policies is another well-researched focus area. Research has shown that privacy policies are hard to comprehend by users [23] and proposals have been made to simplify their understanding [24]. Cranor et al. [5] perform a large-scale study of privacy notices of US financial institutions to highlight a number of concerning practices. Their policy analysis relies on standardized models used for such notices; in contrast, privacy policies in mobile apps follow no such standards making the analysis more challenging. Other approaches [20, 21, 25] have attempted to bridge the gap between users' privacy expectations and app policies. While standardizing privacy policy specification has been attempted [4] with limited success [22], mobile apps' privacy policies have generally failed to adhere to any standards. The findings in our study highlight the need to renew standardization discussion.

6 Conclusion

In this paper, we have presented PolicyLint, a privacy policy analysis tool that conducts natural language processing to identify contradictory sharing and collection practices within privacy policies. PolicyLint reasons about contradictory policy statements that occur at different semantic levels of granularity by auto-generating domain ontologies. We apply PolicyLint on 11,430 privacy policies from popular apps on Google Play and find that around 17.7% of the policies contain logical contradictions and narrowing definitions, with 14.2% containing logical contradictions. Upon deeper inspection, we find a myriad of concerning issues with privacy policies, including misleading presentations and re-defining common terms. PolicyLint's fine-grained extraction techniques and formalization of narrowing definitions and logical contradictions lay the foundation to help ensure the soundness of automated policy analysis and identify potentially deceptive policies.

Acknowledgment

We thank our shepherd, David Evans, and the anonymous reviewers for their valuable comments. This work is supported in part by NSF grants CNS-1513690, CNS-1513939, and CCF-1816615. 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] 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.

[2] Travis D. Breaux and Ashwini Rao. Formal Analysis of Privacy Requirements Specifications for Multi-tier Applications. In *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2013.

[3] Federal Trade Commission. Privacy Online: Fair Information Practices in the Electronic Marketplace: A Federal Trade Commission Report to Congress, May 2000.

[4] Lorrie Faith Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, and Joseph Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. *W3C Recommendation*, 16, April 2002.

[5] Lorrie Faith Cranor, Pedro Giovanni Leon, and Blase Ur. A Large-Scale Evaluation of US Financial Institutions' Standardized Privacy Notices. *ACM Transactions on the Web*, 2016.

[6] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An Empirical Study of Cryptographic Misuse in Android Applications. In *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security (CCS)*, 2013.

[7] 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.

[8] David Evans. Annotation-Assisted Lightweight Static Checking. In *Proceedings of the International Workshop on Automated Program Analysis, Testing and Verification*, 2000.

[9] David Evans, John Guttag, Jim Horning, , and Yang Meng Tan. LCLint: A Tool for Using Specifications to Check Code. In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, 1994.

[10] David Evans and David Larochelle. Statically Detecting Likely Buffer Overflow Vulnerabilities. In *Proceedings of the USENIX Security Symposium*, 2001.

[11] David Evans and David Larochelle. Improving Security Using Extensible Lightweight Static Analysis. *IEEE Software*, January 2002.

[12] Morgan C. Evans, Jaspreet Bhatia, Sudarshan Wadkar, and Travis D. Breaux. An Evaluation of Constituency-based Hyponymy Extraction from Privacy Policies. In *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2017.

[13] Federal Trade Commission Act: Section 5: Unfair or Deceptive Acts or Practices. https://www.federalreserve.gov/boarddocs/supmanual/cch/ftca.pdf.

[14] 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.

[15] 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.

[16] Marti A. Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. In *Proceedings of the Conference on Computational Linguistics (COLING)*, 1992.

[17] Matthew Honnibal and Ines Montani. spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks, and Incremental Parsing, 2017.

[18] Mitra Bokaei Hosseini, Travis D. Breaux, and Jianwei Niu. Inferring Ontology Fragments from Semantic Role Typing of Lexical Variants. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, 2018.

[19] S. C. Johnson. Lint, a C Program Checker. In *Computer Science Technical Report*, pages 78–1273, 1978.

[20] Jialiu Lin, Norman Sadeh, and Jason I. Hong. Modeling Users' Mobile App Privacy Preferences: Restoring Usability in a Sea of Permission Settings. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, 2014.

[21] Fei Liu, Rohan Ramanath, Norman Sadeh, and Noah A. Smith. A Step Towards Usable Privacy Policy: Automatic Alignment of Privacy Statements. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, 2014.

[22] Aditya Marella, Chao Pan, Ziwei Hu, Florian Schaub, Blase Ur, and Lorrie Faith Cranor. Assessing Privacy Awareness from Browser Plugins. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, 2014.

[23] Aleecia M. McDonald and Lorrie Faith Cranor. The Cost of Reading Privacy Policies. *I/S Journal of Law and Policy for the Information Society (ISJLP)*, 4, 2008.

[24] Thomas B. Norton. Crowdsourcing Privacy Policy Interpretation. In *Proceedings of the Research Conference on Communications, Information, and Internet Policy (TPRC)*, 2015.

[25] Ashwini Rao, Florian Schaub, Norman Sadeh, Alessandro Acquisti, and Ruogu Kang. Expecting the Unexpected: Understanding Mismatched Privacy Expectations Online. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, 2016.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

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

[32] 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.

A Preprocessing Privacy Policies

Privacy policies are commonly made available via a link on Google Play to the developer's website and hosted in HTML. Most NLP parsers expect plaintext input; therefore, PolicyLint begins by converting the HTML privacy policy into plaintext. We next describe how PolicyLint achieves this conversion.
Removing Non-relevant and Non-displayed Text: Privacy policies are frequently embedded as main content on a webpage containing navigational elements and other non-relevant text. Additionally, non-displayed text should also be stripped from the HTML, as we want to analyze what is actually displayed to users. PolicyLint extracts the privacy policy portion of the webpage by iterating over the elements in the HTML document. To remove non-relevant text, PolicyLint strips comment, `style`, `script`, `nav`, and `video` HTML tags. PolicyLint also strips HTML links containing phrases commonly used for page navigation (e.g., "learn more," "back to top," "return to top"). Finally, PolicyLint removes HTML `span` and `div` tags using the "`display:none`" style attribute.
Converting HTML to Flat Plaintext Documents: Certain HTML elements, such as pop-up items, result in a non-flat structure. When flattening the HTML documents, PolicyLint must ensure that the plaintext document has a formatting style similar to the text displayed on the webpage (e.g., same paragraph and sentence breaks). PolicyLint handles pop-up elements by relocating the text within the pop-up element to the end of the document. Pop-up elements often provide additional context, explanation, clarification, or a definition of a term. Therefore, relocating these elements should not have a significant effect on processing the referencing paragraph. To ensure that the formatting style is maintained, PolicyLint converts the HTML document to markdown using `html2text`.
Merging Formatted Lists: Formatted lists within text can cause NLP parsers to incorrectly detect sentence breaks or incorrectly tag parts-of-speech and typed dependencies. These parsing errors can negatively impact the semantic reasoning of sentences. Therefore, PolicyLint merges the text within list items with the preceding clauses before the list begins. PolicyLint also uses a set of heuristics for nesting list structures to ensure that list items propagate to the correct clause.

PolicyLint merges formatted lists in two phases. The first phase occurs before the aforementioned conversion to markdown. In this phase, PolicyLint iterates over HTML elements using list-related HTML tags (i.e., `ol`, `ul`, `li`) to annotate list structures and nesting depth. The second phase occurs after the conversion to markdown. In this phase, PolicyLint searches for paragraphs ending in a colon where the next sentence is a list item (e.g., starts with bullets, roman numerals, formatted numbers, or contains annotations from the first phase). It then forms complete sentences by merging the list item text with the preceding text.

PolicyLint iterates over the paragraphs in the markdown document to find those that end in a colon. For each paragraph that ends in a colon, PolicyLint checks whether the proceeding paragraph is a list item. If the line of text is a list item, PolicyLint creates a new paragraph by appending the list item text to the preceding text that ends with the colon. If the list item ends in another colon, PolicyLint repeats the same preceding process but by prepending the nested list items to the new paragraph created in the last step. PolicyLint then leverages the symbols that denote list items to predict the next list item's expected symbol, which is useful for detecting boundaries of nested lists. For example, if the current list item starts with "(1)," then we would expect the next list item to start with "(2)." If the item symbol matches the expected symbol, PolicyLint merges the list item text as discussed above and continues this process. If the item symbol does not match the expected symbol, PolicyLint stops this process and returns.
Final Processing: The final step converts markdown to plaintext. PolicyLint normalizes Unicode characters and strips markdown formatting, such as header tags, bullet points, list item numbering, and other format characters. PolicyLint then uses `langid` to determine whether the majority of the document is written in English. If not, PolicyLint discards the document. If so, PolicyLint outputs the plaintext document.

B Training Sentence Generation

PolicyLint requires a training set of sharing and collection sentences to learn underlying patterns from in order to iden-

Table 8: Sentence Generation Templates

1	*ENT* may *VERB_PRESENT DATA*	We may share your personal information.
2	We may *VERB_PRESENT DATA PREP ENT*	We may share your personal information with advertisers.
3	We may *VERB_PRESENT ENT DATA*	We may send advertisers your personal information.
4	We may *VERB_PRESENT PREP ENT DATA*	We may share with advertisers your personal information.
5	*DATA* may be *VERB_PAST PREP ENT*	Personal information may be shared with advertisers.
6	*DATA* may be *VERB_PAST*	Personal information may be shared.
7	*DATA* may be *VERB_PAST* by *ENT*	Personal information may be shared by advertisers.
8	We may choose to *VERB_PRESENT DATA*	We may choose to share personal information.
9	We may choose to *VERB_PRESENT DATA PREP ENT*	We may choose to share personal information with advertisers.
10	You may be required by us to *VERB_PRESENT DATA*	You may be required by us to share personal information.
11	You may be required by us to *VERB_PRESENT DATA PREP ENT*	You may be required by us to share personal information with advertisers.
12	We are requiring you to *VERB_PRESENT DATA*	We are requiring you to share personal information.
13	We are requiring you to *VERB_PRESENT DATA PREP ENT*	We are requiring you to share personal information with advertisers.
14	We require *VERB_PRESENT_PARTIC DATA*	We require sharing personal information.
15	We require *VERB_PRESENT_PARTIC DATA PREP ENT*	We require sharing personal information with advertisers.
16	We may *VERB_PRESENT ENT* with *DATA*	We may provide advertisers with your personal information.

tify "unseen" sharing and collection sentences. As it is tedious to manually select a set of sharing and collection sentences with diverse grammatical structures, we opt to auto-generate the training sentences instead. Note that auto-generating sentences does not adversely impact the extensibility of PolicyLint, as adding a new pattern is as simple as feeding PolicyLint a new sentence for reflecting this new pattern. To identify the templates, we use our domain expertise to identify different sentence compositions that could describe sharing and collection sentences. We identify 16 sentence templates, as shown in Table 8.

To fill the templates, we substitute an entity (*ENT*), data object (*DATA*), the correct tense of an SoC verb (*VERB_PRESENT*, *VERB_PAST*, *VERB_PRESENT_PARTICIPLE*), and a preposition that describes with *whom* the sharing occurs for sharing verbs (*PREP*). We begin by identifying the present tense, past tense, and present participle forms of all of the SoC verbs (e.g., "share," "shared," "sharing," respectively). We then identify common prepositions for each of the sharing verbs that describe with *whom* the sharing occurs. For example, for the terms *share, trade, and exchange*, the preposition is "with" and for the terms *sell, transfer, distribute, disclose, rent, report, transmit, send, give, provide*, the preposition is "to."

For each template, we fill the placeholders accordingly. If the template has a placeholder for prepositions (*PREP*) and the verb is a collect verb, we skip the template. We also skip the templates for "send," "give," and "provide" if they do not contain a placeholder for a preposition (T1, T6, T8, T10, T12, T14), as those template sentences do not make sense without specifying to *whom* the data is being sent/given/provided. We set *DATA* to the phrases "your personal information" and "your personal information, demographic information, and financial information." Similarly, we set *ENT* to the phrases "advertiser" and "advertisers, analytics providers, and our business partners." Note that we include conjuncts of the *DATA* and *ENT* placeholders to account for deviations in the parse tree due to syntactic ambiguity (i.e., a sentence can have multiple interpretations). Therefore, we generate two sentences for each template: one with a singular *DATA* and *ENT*, and second with the plural *DATA* and plural *ENT*. In total, we generate 560 sentences, which are used by PolicyLint to learn patterns to identify sharing and collection sentences.

C Policy Statement Extraction

For the 9% of the policies from which PolicyLint does not extract policy statements, we randomly select 100 policies to explore why this situation occurs. We find that 88% (88/100) are due to an insufficient crawling strategy (i.e., privacy policy links being pointed at home pages, navigation pages, or 404 error pages). Among the remaining 12 policies, the links in 3 policies point at a PDF privacy policy while PolicyLint handles only HTML, and 2 policies are not written in English but are not caught by our language classifier. Finally, 7 policies are due to errors in extracting policy statements, such as incomplete verb lists (3 cases), tabular format policies (1 case), and policies for describing permission uses (3 cases).

D Email Template

Subject: Contradictory Privacy Policy Statements in <APP_NAME>
To Whom It May Concern:
We are a team of security researchers from the WSPR Lab in the Department of Computer Science at North Carolina State University. We created a tool to analyze privacy policies and found that the privacy policy for your "<APP_NAME>" application (<PACKAGE>) that we downloaded in September 2017 may contain the following potential contradictory statements:
(1) The following statements claim that personal information is both collected and not collected.
(A) <CONTRADICTORY_SENTENCE_1>
(B) <CONTRADICTORY_SENTENCE_2>
...
Do you believe that these are contradictory statements? Any additional information that you may have to clarify this policy would be extremely helpful. If you have any questions or concerns, please feel free to contact us, preferably by February 11th.
Thank you for your time!
Best Regards,

<EMAIL_SIGNATURE>

Figure 6: Email Template