PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play

Abstract

Privacy policies are the primary mechanism by which mobile applications inform users about data collection and sharing practices. These policies are long, complex legal documents, and as a result it is difficult for users, regulators, or application analysis tools to identify at scale objectionable or problematic practices. In this paper, we present PolicyLint, a privacy policy analysis tool inspired by lightweight static analysis tools like `lint`. PolicyLint automatically generates ontologies capturing semantic relationships between data types and entities from a large corpus of privacy policies. It then uses sentence-level natural language processing to capture both positive and negative statements of data collection and sharing. We use PolicyLint to analyze 11,430 apps for contradictions in their sharing and collection statements. In this novel study, we find 2,060 policies that are potentially contradictory. We manually verify 510 contradictions, finding many concerning trends, including 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 derive sensitive information. In so doing, PolicyLint significantly advances automated policy analysis.

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. Prior work has demonstrated that apps regularly transmit this information to first or third parties []. This data collection and sharing is often considered (legally) acceptable if it is described in the privacy policy for the application. These 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 if app developers adhere to privacy policies, to help app stores and other analysts identify privacy violations, or help end users choose more-privacy-friendly apps.

Recent work has begun studying privacy policies and mobile apps to address these questions [22, 25, 28, 24]. However, these efforts have suffered from three limitations in policy language analysis. First, privacy policies referto information at different semantic granularities. Prior work has tackled this issue by crowdsourcing data object ontologies [22, 24],[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") [28] or detecting only verb modifiers [25] while neglecting the more-complicated statements (e.g., "will share X except Y") that are common in privacy policies. Finally, mixing positive and negative statements or varying levels of specificity may lead to intentional or accidental contradictions. This issue has not been studied in the context of a single policy, though one project has compared app policies with the policies of their included libraries [25].

In this paper, we address the preceding limitations by presenting PolicyLint, a tool designed to automatically analyze the content of privacy policies. PolicyLint is inspired by other security `lint` tools [7, 6, 8, 9, 5, 14], which analyze code for indicators of potential bugs or other programming errors. Like any static approach, not every `lint` finding is necessarily an error. 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 intentially vague, ambiguous, or even deliberately misleading even for human interpretation. Despite these fundamental challenges, PolicyLint provides a mechanism to condense a long and compicated policy into a small set of candidate issues that would interest a human or algorithmic analysis. We note that a wide body of prior research has analyzed the content of privacy policies for aspects like coverage and clarity [1, 11, 10]. While PolicyLint could cer-

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

tainly assist in answering similar questions, in this paper, we focus on a challenging and important, yet little-studied question: *"do privacy policies contain contradictions?"* Such contradictions would make policies unclear, confusing both humans and any automated system that relied on interpreting the policy. PolicyLint is the first tool have the sophistication necessary to reason about both negative sentiments and statements covering varying levels of specificity, which is necessary for uncovering contradictions.

This paper makes the following contributions:

- **Automated generation of ontologies from privacy policies.** PolicyLint uses an expanded set of Hearst patterns [**?**] 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 crowdsourcing.

- **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 practices** We formally model 9 types of contradictions that result from semantic relationships between terms, providing an algorithmic method to detect contradictory policy statements. In a study of 11,430 privacy policies from mobile apps, we are the first to find that such contradictions are *rampant*, affecting 18% of policies.

- **Manual analysis of contradictions to identify trends** The high rate of policy contradictions is surprising. We manually reviewed 510 contradictions across 260 policies, finding that many contradictions are indeed indicators of misleading or problematic practices. These include making broad claims to protect personal information early in a policy, yet later carving out exceptions for data that 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.

This paper is organized as follows: Section 2 describes PolicyLint's design. Section 3 automatically derives ontologies describing data types and data-handling entities. Section 4 extracts discrete policy statements from privacy policies. Section 5 formally defines privacy-policy contradictions. Section 6 reports on our empirical study using PolicyLint. Section 7 describes related work. Section 8 concludes.

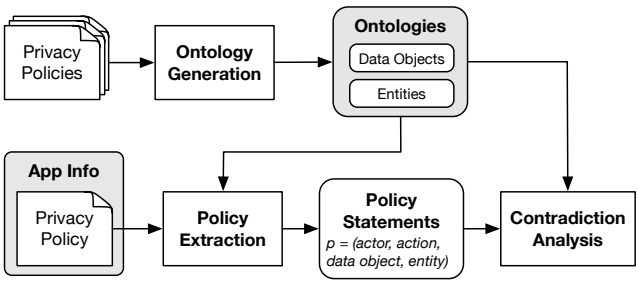

Figure 1: Overview of PolicyLint

2 PolicyLint

PolicyLint seeks to identify contradictions within individual privacy policies for software. PolicyLint provides privacy *warnings* based on contradictory sharing and collects 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 contradicting policy statements when considered in the most conservative interpretation (i.e., context-free). 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 application (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 application does not collect any personal information from users. 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 part is a clear contradiction to the earlier part, as email address is consider personal information. As discussed in detail in Section 6, the cause for this underlying contradiction is that the developer does not consider email address as personal information.

To our knowledge, our work is the first to consider 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 [22, 24] uses ontologies to capture subsumptive relationships between terms; however, such ontologies are crowdsourced, leaving concerns of comprehensiveness and scalability.

- *Privacy policies include negative sharing and collection statements.* Most prior work operates at paragraph level and cannot capture negative sharing statements. Prior work that does capture negative statements [28, 25]

misses complex statements (e.g., "will share personal information except your email address).

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 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 [12], which we have extended for our domain.

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

3 Ontology Generation

The goal of the 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 privacy policy text, 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 objects.

PolicyLint uses a semi-automated and data-driven approach 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 relations 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 of seed words as input and generates data-object/entity ontologies using the subsumptive relations discovered in the prior step. It iteratively adds relations to the ontology until a fixed point is reached. We now describe this process in detail.

3.1 NER Domain Adaptation

To identify subsumptive relations 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 approach of named-entitity recognition (NER) to label data objects and entities within sentences. Prior research [22, 24, 25, 28] proposed keyphrase-based approaches for identifying data objects. However, key-phrase approaches are less versatile in practice: they cannot handle term ambiguity and variability, and they can identify only terms defined in their pre-defined list. For example, the term "internet service provider" can be both a data object and an entity, which cannot be differentiated by keyphrase-based approaches. 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 that describe entities, e.g., "advertisers"). Training an NER model from scratch is a time-consuming process due to the large amount of training data required to achieve a reasonable performance. Therefore, PolicyLint starts from an existing NER model and updates it using annotated examples of training data from our problem domain. Specifically, PolicyLint adopts Spacy's NER engine [13], which uses deep convolutional neural networks. We adapt the *en_core_web_lg* model to the domain of privacy policies.

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 lexico-syntactic patterns described in Section 3.2, 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 dataset, shuffling at each epoch, and using minibatch training with a batch size of 4. To perform the domain adaptation, the current model attempts to predict the NER labels for each word in the sentence and will adjust the synaptic weights in the neural network accordingly if the prediction does not match the annotations. We stop making passes over the training data when the loss rate

Table 1: Lexicosyntatic patterns for subsumptive relations

#	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 $[or\|and]$ *other* $Y_1, Y_2, \ldots Y_n$
H4	*X, including* $Y_1, Y_2, \ldots Y_n$
H5	*X, especially* $Y_1, Y_2, \ldots Y_n$
C1	*X,* $[e.g.\|i.e.]$, $Y_1, Y_2, \ldots Y_n$
C2	*X (* $[e.g.\|i.e.]$, $Y_1, Y_2, \ldots Y_n$*)*
C3	*X, for example,* $Y_1, Y_2, \ldots Y_n$
C4	*X, which may include* $Y_1, Y_2, \ldots Y_n$

* H* = Hearst Pattern; C* = Custom Pattern

begins to converge. We annotate an additional 100 randomly selected sentences as holdout data for testing the model. Table 5 (Appendix C) shows the performance of the NER model before and after domain adaptation for our holdout dataset. PolicyLint achieves 82.2% and 86.8% precision for identifying data objects and entities, respectively.

3.2 Subsumptive Relation Extraction

PolicyLint uses a set of 9 lexicosyntactic patterns to discover subsumptive relations within sentences, as shown in Table 1. The first five are Hearst Patterns [12], and the last four are custom deviations based observations of text in privacy policies. For each pattern, PolicyLint ensures 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, $Y_1, Y_2, \cdots, Y_n$ is a conjunctive noun phrase, 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 complete set of extracted relations, PolicyLint normalizes the relations by lemmatizing the text and substituting terms with their synonym. For example, consider that we know "blood sugar levels" is a synonym for "blood glucose level." Lemmatization turns"blood sugar levels" into "blood sugar level", and synonym substituion turns it into "blood glucose level". To identify synonyms, we output the non-terminal (i.e., X value of the Hearst patterns) data objects and entities in the subsumptive relations. 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 dump out all of the data objects and entities labeled from all the 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 our domain knowledge to identify that "latitude and longitude" is a synonym of "location", output out the terms that contain those words, and manually identify synonyms.

3.3 Ontology Construction

PolicyLint generates ontologies by combining the subsumptive relations extracted from policies with a set of seed terms (Table 6, Appendex D). For each ontology, PolicyLint iterates through the seeds, selecting relations that contain it. PolicyLint then expands the term list from the relations in that iteration. PolicyLint continues iterating over the relations until no new relations are added to the ontology. If there exists any inconsistent relation where X is subsumed under Y and Y is subsumed under X, PolicyLint uses the relation 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 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 relations and ensures that the destination node has more than one in-edge to ensure that a new root node is not created.

4 Policy Statement Extraction

The goal of policy statement extraction is to identify a set of 4-tuples, (actor, action, data object, entity), as described in Section 5. PolicyLint processes privacy policy text at sentence-level granularity to ensure precise modeling of negative sentiment policy statements (e.g., negated verbs) and exception clauses. The following example includes both a negated verb and an exception clause.

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

PolicyLint extracts policy statements from the 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 SoC verbs). First, PolicyLint transforms a sentence's dependency-based parse tree into a concise format that captures generalizations of the relations between named-entities (i.e., data objects and entities) and the SoC verbs in the sentence. We call this sentence structure a data and entity dependency (DED) tree. Second, PolicyLint extracts DED trees from positive examples of sentences that describe sharing or collection practices. These DED trees represent known patterns of sharing and collection phrases. Finally, PolicyLint uses those patterns to determine whether "unseen" sentences describe a sharing or collection practice. If so, PolicyLint extracts policy statements based on positive matches. The remainder of this section describes these steps in detail.

4.1 DED Tree Construction

The goal of constructing the data and entity dependency (DED) trees is to extract a concise representation of the grammatical relations between the data objects, entities, and the verbs that represent sharing or collection (SoC verbs). The main intuition behind constructing these trees is to allow PolicyLint to infer semantics of the sentence based on the grammatical relations 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, and SoC verbs, and performs a set of simplifications to generalize the representation. The transformation for Example 2 is shown in Figure 2.

To construct DED trees, PolicyLint parses a sentence and using its custom-trained NER model to label data objects and entities within the sentence (Section 3). PolicyLint merges noun phrases and iterates over sentence tokens to label SoC verbs by ensuring the PoS tag of the token is a verb and the lemma of the verb is in PolicyLint's manually curated list of terms (Table 7, Appendex D). 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 above.

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 descendent verb nodes in three situations. First, if a descendent 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 descendent 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 descendent 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", it 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", "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 subtrees, 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., $\{\text{share}\}$ + $\{\text{collect}\}$ = $\{\text{share, collect}\}$. 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 subtree contains at least one other node labeled as an SoC verb, data object, or entity. If the node's subtree 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 subtree and that it meets the conditions above for an SoC verb. Similarly, if these conditions are not met, PolicyLint also removes the subtree rooted at that non-labeled verb node. For example, this pruning step causes the subtree rooted at the verb "register" to be removed in Figure 2.

4.2 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. To begin, we feed 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. Note that the example sentences are auto-generated from a set of 16 sentence templates, as described in Appendix B. We choose to auto-generate the sentences, as manually selecting a set of sentences with diverse grammatical structures is a tedious process. Doing so does not adversely impact the extensibility of PolicyLint, as adding a new pattern is as simple

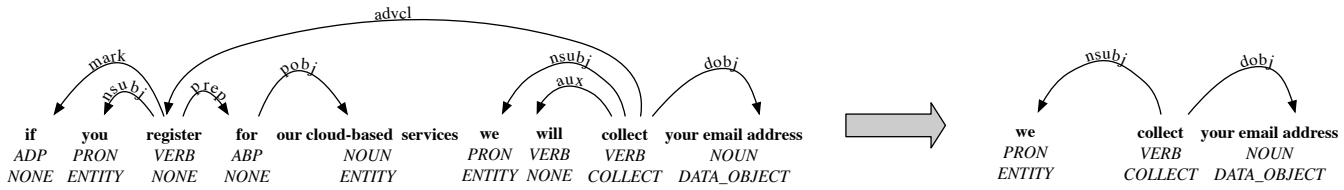

Figure 2: Transformation of Example 2 from its dependency-based parse tree to its DED tree.

as feeding PolicyLint the new sentence.

PolicyLint iterates sentences 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 DED tree to the DED trees of the known patterns. A pattern is matched if (1) the sentence DED tree's label types are equivalent to the pattern DED tree (e.g., {*entity*, *SoC_verb*, *data*}), and (2) the known pattern DED tree is a subtree of the sentence 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 subtree match, PolicyLint records the nodes in the subtree match and continues the process until either (1) each pattern is checked, or (2) the entire DED tree has been covered by prior subtree matches. If at least one subtree match is found, PolicyLint identifies the sentence as a potential SoC sentence and begins extracting the policy statement tuple.

4.3 Policy Extraction

The goal of the policy extraction phase is to transform the DED tree into a (*actor*, *action*, *data object*, *entity*) tuple for a CPS (Section 5.1). PolicyLint performs policy extraction starting with the SoC nodes present in the subtree matches. If multiple SoC nodes exist in the subtree matches, multiple CPSes are generated. However, multiple subtree matches over the same SoC node will only result in the generation of one CPS. The SoC determines the action (e.g., collect, not_collect). The sentiment of the action is determined based on whether the node is labeled with positive or negative sentiment, as discussed in Section 4.1.

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 xcomp or advcl, PolicyLint prioritizes the direct object and prepositional object over the subject. 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 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 a 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 Simplification: 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]) expands to (we, share, location, Google), (we, share, location, Facebook), (we, share, age, Google), etc.

For each extracted policy statement, PolicyLint creates simplified policy statements, as described in Section 5.1. However, there are two special cases. First, PolicyLint only treats verb lemmas "save" and "store" with a positive sentiment ("not saving" or "not storing" does not mean "not collecting"). Second, PolicyLint ignores policy statements with verb lemma "use" and negative sentiment. This case leads to false positives, as PolicyLint does not extract the collection purpose. For example, "*We do not use personal information for advertising.*" means that personal information is not collected for the specific purpose of advertising.

5 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 application'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.

5.1 Policy Statement Representation

A complete policy statement (CPS) is a concise representation of a data collection or sharing practice from the privacy policy. A CPS can be represented 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 of "We will share your personal information with advertisers" can be represented by the tuple of (we, share, personal information, advertisers).

PolicyLint transforms complete policy statements into a simpler representation to reduce the complexity of formalizing contradictions within a policy. In particular, we simplify policy statements involving the sharing of data (i.e., the CPS *action* is share or not share). We capture sharing as collection using a simplified policy statement (SPS) defined as follows.

Definition 1 (Simplified Policy Statement)**.** *An SPS is a tuple, $p = (e, c, d)$, where d is the data object that the statement is discussing, $c \in \{collect, not\_collect\}$ 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 2). 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 2). 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 2). Finally, a policy for stating that the actor does not share a data object implies that the actor collects the data object, because the policy would likely have not mentioned not sharing data that was never collected (Rule T4, Table 2).

For example, consider we have the following complete policy statements: {(we, share, personal information, advertisers), (we, not share, your email address, analytics providers)}. After applying the transformation rules, we have the following policy statements: {(we, collect, personal information), (advertisers, collect, personal information), (analytics providers, not_collect, your email address), (we, collect, your email address)}.

5.2 Contradiction Types

We model an application'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, collect, demographics) and $p_2 =$ (Google Admob, not_collect, age). If we want to identify contradictions, we need to know that Google AdMob is an advertiser and age is demographic information. These of relationships are commonly referred to as *subsumptive relationships* where a more specific term is subsumed under a more general term (i.e., 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

Table 2: 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

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 in a policy if two policy statements suggest that the entities both may and may not collect a data object. Contradictions can occur at the same or different semantic levels. For example, the most simple 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 relation applies to both entities and data objects. Therefore, there are $3^2 = 9$ types of contradictions. Table 3 lists all 9 types along with intuitive examples.

5.3 Contradiction Identification

PolicyLint uses the contradiction types from Table 3 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 applications), 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 if 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 8 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 PolicyLint ignores entities and data objects in policy statements that are not contained in ontologies, as it cannot reason about those relations. However, if PolicyLint cannot find a direct match for a term in the ontology, it will try to find sub-matches by splitting the term on the coordinating conjunction terms (e.g., "and", "or") and checking for their existence in the ontologies.

Candidate Set Reduction: PolicyLint uses heuristics to prune candidate contradictions that are likely low quaility indicators of underlying problems. To begin, PolicyLint removes contradictions if the policy statements were generated from the same sentence. 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), which causes a *C2* contradiction. PolicyLint ignores same sentence contradictions, as they typically occur to provide clarifications.

PolicyLint removes contradictions that occur based on potentially poor relations discovered in the 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", "material") or entities (e.g., "individual", "public") are also removed. Finally, PolicyLint removes candidate contradictions where the negative sentiment policy statement may be conditioned with age restrictions or based on user choice by searching for common phrases in the sentences that generated 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 contradiction 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 generated them in context of the entire policy and makes a decision.

6 Privacy Study

Our primary motivation for creating PolicyLint was to analyze contradicting 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 applications from September 2017.

Dataset Collection: To select our dataset, we scraped Google

Table 3: Policy Contradictions: $P = \{(e_i, \text{collect}, d_k), (e_j, \text{not\_collect}, d_l)\}$

Rule	Logic	Example	Example Explanation
C1	$e_i \equiv_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(companyX, collect, email address) (companyX, not_collect, email address)	Inconclusive due to exact contradiction
C2	$e_i \equiv_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(companyX, collect, email address) (companyX, not_collect, personal information)	companyX does not collect any personal information other than your email address
C3	$e_i \equiv_\varepsilon e_j \wedge d_k \sqsupset_\delta d_l$	(companyX, collect, personal information) (companyX, not_collect, email address)	companyX may collect any personal information other than your name
C4	$e_i \sqsubset_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(companyX, collect, email address) (advertiser, not_collect, email address)	No advertiser may collect your email address except for companyX
C5	$e_i \sqsubset_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(companyX, collect, email address) (advertiser, not_collect, personal information)	No advertiser may collect any personal information except companyX may collect your email address
C6	$e_i \sqsubset_\varepsilon e_j \wedge d_k \sqsupset_\delta d_l$	(companyX, collect, personal information) (advertiser, not_collect, email address)	companyX may collect any personal information except for your email address
C7	$e_i \sqsupset_\varepsilon e_j \wedge d_k \equiv_\delta d_l$	(advertiser, collect, email address) (companyX, not _collect, email address)	Any entity that is an advertiser except for companyX may collect your email address
C8	$e_i \sqsupset_\varepsilon e_j \wedge d_k \sqsubset_\delta d_l$	(advertiser, collect, email address) (companyX, not_collect, personal information)	Any advertiser including companyX may collect your email address
C9	$e_i \sqsupset_\varepsilon e_j \wedge d_k \sqsupset_\delta d_l$	(advertiser, collect, personal information) (companyX, not_collect, email address)	companyX may collect any personal information except for your email address

Play for the privacy policy links for up to the top 500 free applications across Google Play's 35 application categories in September 2017. We used use the Selenium WebDriver in a headless Google Chrome browser to allow for the execution of dynamic content (e.g., Javascript). We exclude applications that did not have a privacy policy link on Google Play, those whose privacy policy pages were unreachable at the time of data collection, and privacy policies where the majority of the document was not written in English, as discussed in Appendix A. We converted the HTML privacy policies to plaintext documents. Our final dataset consists of 11,430 privacy policies.

6.1 General Policy Characteristics

PolicyLint extracted policy statements from 91% of the policies in our dataset (10,397/11,430). From those policies, PolicyLint extracted 438,667 policy statements from 177,169 sentences that PolicyLint identified as a sharing or collection sentence. Of those policy statements, 32,876 had negative sentiment and 405,789 had positive sentiment. In particular, 60.5% (6,912/11,430) of policies had at least one negative sentiment policy statement and 89.6% (10,239/11,430) of policies had at least one positive sentiment policy statement.
Finding 1: *Policies frequently contain negative sentiment policy statements that discuss broad categories of data.* For the 60.5% of policies with at least one negative sentiment policy statements, the data object "personal information" appeared in 67.7% of those policies (4,681/6,912). This demonstrates the importance of handling negative policy statements, as around 41.0% of the policies contain a negative sentiment policy statement that claims that a broad type of data (i.e., "personal information") is not collected. Further, we measured the distance from the negation (i.e., "not") to the verb that they modify, and found that the 28.2% (3,234/11,430) of policies have a distance greater than one word away. This calls into questions prior work [22, 24] that only assume positive sentiment when considering sharing and collection statements when verifying behavior-to-policy compliance, as they could be incorrect up to 60.5% of the time when reasoning over sharing and collection statements. Further, approaches that handle negations using bigrams [28] would have failed to reason about 28.2% of the policies.

6.2 Candidate Contradictions

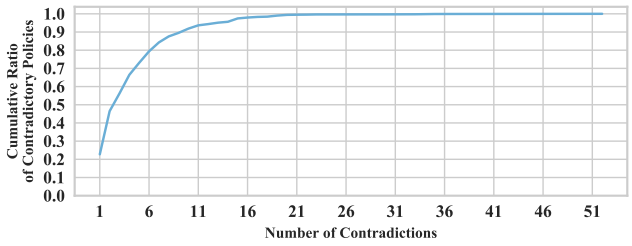

Figure 3: CDF of Contradictory Policy statements: Around 50% of the policies containing one or more contradictions have 3 or fewer unique candidate contradictions.

Based on PolicyLint's fine-grained policy statement extraction, we found that 59.1% (6,754/11,430) of the policies were candidates for contradiction analysis, as they contain at least one positive and one negative sentiment policy statement. There were 13,871 and 129,575 policy statements among these with negative and positive sentiment, respectively.
Finding 2: *18.0% of privacy policies contain candidate contradictions.* PolicyLint identified 18,457 candidate contradictions across around 18.0% (2,055/11,430) of policies. Figure 3 shows slightly more than half (56.3%) of contradictory policies have 3 or fewer unique candidate contradictions. The relatively low number of candidate contradictions per policy indicates that manual validation is feasible. Further, roughly

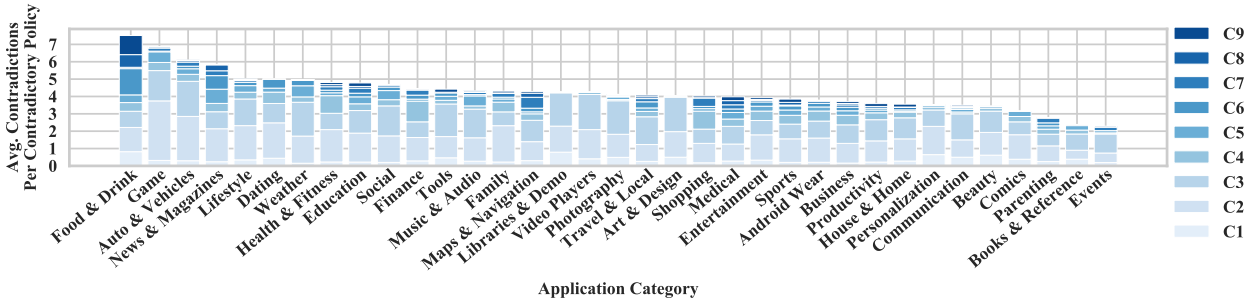

Figure 4: Average Number of Policy Contradictions Per Application Category.

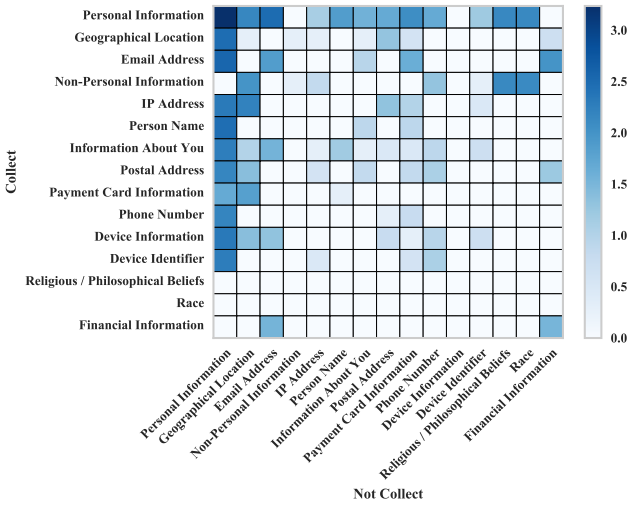

Figure 5: Log 10 Frequency of Data Type Pairs in Contradictions: Negative sentiment policy statements that discuss broad categories of data are problematic.

4 in 5 policies are not contradictory, indicating writing non-contradictory policies is possible.

Finding 3: *Contradiction prevalence and frequency does not substantially vary across Google Play app categories.* We plotted the ratio of policies containing candidate contradictions for apps in each Google Play category to analyze whether policies from certain categories were more contradictory than others. Note that we exclude the graph in the paper due to space constraints. The categories with policies least and most prone to contradiction are *Beauty* and *Events*, respectively. However, when analyzing the policies within those categories, we found that their means were being skewed by contradictory policies for applications by the same developer. When recomputing the average, these categories followed the general trend. Contradictory policies accompany 10-to-30% of apps across all categories. We find that contradictory policies are not substantially more prevalent in particular categories of apps, but instead occur consistently in apps from every category. In a similar analysis, we found that contradiction prevalence does not substantially vary by download count as well.

Figure 4 displays the average number of candidate contradictions for policies containing one or more candidate contradictions. We found that contradiction frequency for contradictory policies did not substantially vary across Google Play app categories. Initial analysis indicated contradictory policies for apps in the *Food & Drink* category contain over seven contradictions on average. Further analysis revealed this is due to policies with 15 unique contradictions in 31 apps produced by the same developer. Excluding the outliers these brings the category's average to 3.44 contradictions per app, which fits the trend of the rest of the categories. This may indicate that poor policies are linked to problematic developers. Similar analysis on *GAMES*, *Auto & Vehicles*, *News & Magazines* produced similar results. We find that the number of contradictions per policy is roughly equivalent across application categories, which means that one application 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 contradicting policy statements in the leftmost column are most problematic. This column represents contradictions that discuss not collecting broad types of data and collecting more specific data types. As we demonstrate in Section 6.3, they can lead to a myriad of problems when trying to interpret the policy including making interpretation ambiguous in certain cases.

The topmost row corresponds to cases where broad information is stated to be collected, and specific data types are then stated to not be collected. This is a common occurrence, yet it may not necessarily be an unwanted property of policies, because saying a broad data type is collected does not necessarily imply that every specific subtype of data is collected. There may be clearer ways for policy writers to convey this information, such as explicitly stating all of the data types collected and shared. However, policies that explicitly narrow the scope of their data sharing and collection practices can be seen as more desirable in contrast with policies that just disclose practices over broad categories of data.

6.3 Validating Policy Contradictions

In this section, we describe the findings from our validation of candidate contradictions. Due to resource constraints, we did not validate all of 18,457 candidate contradictions from the 2,060 policies. Instead, we narrowed the scope of our study by choosing groups of candidate contradictions based the data objects involved by analyzing Figure 5. We mainly limited our scope to candidate contradictions that discussed not collecting "personal information" and collecting data that is subsumed under or equivalent to personal information (i.e., contradiction types C1, C2, C5, C8). In paricular, we explore candidate contradictions that discuss not collecting "personal information" and collecting "email address," "device identifier," or "personal information."

Note that while the majority of our study is focused on personal information, we also explored two other groups of candidate contradictions that caught our attention when analyzing the heatmap. In particular, we were surpised to find that our ontology captured contradictions between data that can be *derived* from the other. The groups of candidate contradictions that we validated for *derived data* were candidate contradictions that discussed: (1) not collecting "geographic location" and collecting "IP address"; and (2) not collecting "postal address" and collecting "geographic location."

To validate candidate contradiction, we begin by reading through the sentences that generated each policy statement for that candidate contradiction to ensure correctness of our policy statement extraction. If there was 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 occurred if possible and record any obversations about the policy.

6.3.1 Personal Information and Email Addresses

For candidate contradictions with negative statements about "personal information" and with positive statements about "email address," we found 618 candidate contradictions across 333 policies (C2, C5, C8) We validated 204 candidate contradictions from 120 policies. We found 5 candidate contradictions were false positives due to inaccuracies labeling data objects by the NER model. From the 199 remaining candidate contradictions across 118 policies, we had the following main findings.Note that when considering 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 5: *Policies are stating that certain types of common personally identifiable information, such as email addresses, as non-personally identifiable.* When validating 14 candidate contradictions, we found 14 policies that explicitly state that they do NOT consider email address as personally identifiable information. 11 of those policies were from apps released the same developer (OmniDroid) where the most popular app in the set (com.omniluxtrade.allrecipes) has over 1M+ downloads. OmniDroid's privacy explicitly lists email address when defining non-personally identifiable information. The remaining 3 policies belong to another app developer, PlayToddlers. The app in our dataset are explicitly targeted towards 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 6: *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 found 104 policies broadly make blanket statements that personal information is not collected in one part of the policy and then directly contradict their prior statements by disclosing that they collect email addresses. We found 69 of those policies (127 validated contradictions) state that they do not collect personal information, but later state 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.

20 of those policies that explicitly defined email as personal information, but contradicted themselves, are from apps by the same organization (emoji-keyboard.com). The most popular app in that group had 50M+ downloads (emoji.keyboard.emoticonkeyboard). The following two sentences were 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 is clearly a contradictory statement and arguably a misleading practice.

A policy for a particular app with 1M+ downloads (com.picediting.haircolorchanger) appear 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 was in the the policy verbatim including the bold formatting: *"**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..."*

Finding 7: *Policies consider hashed email addresses as pseudonymized non-personal information and share it with advertisers.* When validating three candidate contradictions, we found two policies discuss sharing hashed email addresses with third parties, such as advertisers. One candidate contradictions was a false positive due to misclassifying a sentence discussing opt-out choices as a sharing or collection sentence. The other policy belonged 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.

Finding 8: *Services that auto-generate template-based policies for app developers are producing contradictory policies.* During our validation of the policies in the prior findings, we noticed that many policies had similar structural compositions and contained a lot of the same language in paragraphs. When validating 78 candidate contradictions, we found 59 contradictory policies that were automatically generated or used templates. Identical policy statements from various developers suggested that some policies may be generated automatically or acquired from a template. We investigated these cases and identified 59 policies that used 3 unique templates. We check that these were not policies for apps created by the developers or organization. The problems that the general templates were causing are discussed in Findings 6 and 10.This demonstrates that poor policy generators can be a contributing factor for numerous contradictory policies.

6.3.2 Personal Information and Device Identifiers

For the candidate contradictions with negative statements about "personal information" and with positive statements about "device identifiers," we found 234 candidate contradictions across 155 policies. We investigated this group of candidate contradictions as there are differing regulations across countries on whether device identifiers are consider personal information. For example, a court case from New York (Robinson v. Disney Online) ruled that device identifiers are not personal information. However, the GDPR defines device identifiers as personal information. Therefore, our goal was to check whether policies were complying to the more strict GDPR definition of personal information or to the US definition, as this could hint towards problems with complying to regulations across country boundaries. In total, we validated 10 candidate contradictions across 9 policies.

Finding 9: *Policies are considering device identifiers as non-personal information, which raises concerns regarding globalization of their policies.* When validating 10 candidate contradictions, we found 9 policies that state that they do not collect personal information, but later state they they collect device identifiers. We find that classification of device identifiers vary across policies. In particular, we found 4 policies that explicitly describe device identifiers as non-personal information. The most popular app is Tango (com.sgiggle.production), which boasts about 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 there current privacy policy still contains this statement, it may hint that they may not be GDPR compliant.

6.3.3 Personal Information and Personal Information

For the candidate contradictions where the data type of the negative sentiment policy statement is "personal information" and the data type of the positive sentiment policy statement is "personal information", we found 5100 candidate contradictions across 1061 policies. We validate 254 candidate contradictions across 153 policies.

Finding 10: *Policies directly contradict themselves.* When validating the 254 candidate contradiction, we found that the 153 policies directly contradicted themselves on their data practices on "personal information". For example, the policy for an application 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.

6.3.4 Derived Data

In this section, we discuss two cases of contradictions that PolicyLint captures between data objects where one data type can be derived from the other. We explore two cases: (1) location from IP address; and (2) postal address from location. In particular, we explore candidate contradictions where the negative policy statements discuss the data that can be derived from the other piece (i.e., location for "location from IP", and

postal address for "postal address from location").

For "location from IP", we found 170 candidate contradictions from 167 policies that represented collecting IP address and not collecting location information. We narrowed down the candidate contradictions by filtering out statements that discuss precise location, as IP address does not provide a precise location. This filtering resulted in 18 candidate contradictions from 18 different policies. We note that 3 candidate contradictions from 3 policies were false positives due to incorrect negation handling. We validated 15 candidate contradictions across 15 different policies for this case.

For "postal address from location", we found 27 candidate contradictions across 20 policies. 5 candidate contradictions were false positives due to sentence misclassification (4) or errors handling negations (1). We validated the remaining 22 candidate contradictions across 17 applications for this case. **Finding 11**: *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 "location from IP", we found that all 15 policies were 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, 2 of the policies discuss that if users disable location services, then location will not be collected. However, they still collect IP addresses regardless of the privacy settings.

When validating 20 candidate contradictions for "postal address from location", we found that 15 policies discussed not collecting postal addresses, but then state that they collect locations. Similar to the above case, postal addresses can be derived from location data (i.e., latitude and logitudes). Again, the argument can be made that they are collecting information precise enough to be considered a postal address, which causes a contradiction. For the other 2 candidate contradictions from 2 policies, it was not clear whether it was actually a contradiction, as they state that they do not collect addresses from the user's address book, which is a limitation of our tool on handling context sensitivity.

7 Related Work

Recent research has increasingly focused on automated analysis of privacy policies. Systems have used NLP for deriving answers to a limited of binary questions [27], from privacy policies, applied topic modeling to reduce ambiguity in privacy policies [23], and used data mining [26] or deep learning [11] models to extract summaries from policies of what and how information is used. Other related works [22, 24] have used *crowdsourced* ontologies for policy analysis. These methods are often limited by lack of accuracy, completeness, and collection complexity. Prior research has also attempted to infer negative statements in privacy policies with limited success. Zimmeck et al. [28] and Yu et al. [25] rely on keyword-based techniques of using bi-grams and verb modifiers, respectively, to detect the negative statements. In contrast to all these previous approaches, our work provides a more comprehensive analysis with a automatically constructed ontology and accounting for negations and exclusions in text.

To the best of our knowledge, we are the first to analyze single policy-level contradictions resulting from the interaction of varying semantic levels with negative statements. However, Yu et al. [25] developed a system that verifies the consistency of an app's privacy policy with the privacy policies of the libraries used by the app.

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 [18] and proposals have been made to simplify their understanding [19, 20]. Cranor et al. [4] performed 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 [15, 16, 21] have attempted to bridge the gap between users' privacy expectations and app policies. SPARCLE [2] follows an alternative approach of privacy compliance by deriving machine-readable operational directives from policies written in natural language. While standardizing privacy policy specification has been attempted [3] with limited success [17], privacy policies for mobile apps have generally failed to adhere to any standards. The concerning findings in our study highlight the need to renew such standardization discussion.

8 Conclusion

This work introduced PolicyLint, a privacy policy analysis tool that uses natural language policy techniques 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 run PolicyLint on 11,430 privacy policies from popular applications on Google Play and find that around 18% of the policies contain contradictions. Upon deeper inspection of the contradictions, we found a myriad of concering issues with privacy policies, such as policies explicitly defining common subtypes of personal information as non-personal information, such as email addresses. PolicyLint can assist this process by automatically reasoning over these problematic areas within policies.

References

[1] BOWERS, J., REAVES, B., SHERMAN, I. N., TRAYNOR, P., AND BUTLER, K. Regulators, Mount Up! Analysis of Privacy Policies for Mobile Money Services. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)* (2017).

[2] BRODIE, C. A., KARAT, C.-M., AND KARAT, J. An Empirical Study of Natural Language Parsing of Privacy Policy Rules Using the SPARCLE Policy Workbench. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)* (2006).

[3] CRANOR, L. F., LANGHEINRICH, M., MARCHIORI, M., PRESLER-MARSHALL, M., AND REAGLE, J. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. *W3C Recommendation 16* (Apr. 2002).

[4] CRANOR, L. F., LEON, P. G., AND UR, B. A Large-Scale Evaluation of US Financial Institutions' Standardized Privacy Notices. *ACM Transactions on the Web (TWEB)* (2016).

[5] EGELE, M., BRUMLEY, D., FRATANTONIO, Y., AND KRUEGEL, C. An empirical study of cryptographic misuse in Android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013).

[6] EVANS, D. Annotation-Assisted Lightweight Static Checking. In *The First International Workshop on Automated Program Analysis, Testing and Verification* (2000).

[7] EVANS, D., GUTTAG, J., HORNING, J., , AND TAN, Y. M. LCLint: A Tool for Using Specifications to Check Code. In *SIGSOFT Symposium on the Foundations of Software Engineering* (1994).

[8] EVANS, D., AND LAROCHELLE, D. Statically Detecting Likely Buffer Overflow Vulnerabilities. In *2001 USENIX Security Symposium* (2001).

[9] EVANS, D., AND LAROCHELLE, D. Improving Security Using Extensible Lightweight Static Analysis. *IEEE Software* (Jan. 2002).

[10] GLUCK, J., SCHAUB, F., FRIEDMAN, A., HABIB, H., SADEH, N., CRANOR, L. F., AND AGARWAL, Y. How short is too short? Implications of length and framing on the effectiveness of privacy notices. In *12th Symposium on Usable Privacy and Security (SOUPS)* (2016), pp. 321–340.

[11] HARKOUS, H., FAWAZ, K., LEBRET, R., SCHAUB, F., SHIN, K. G., AND ABERER, K. Polisis: Automated analysis and presentation of privacy policies using deep learning. In *Proceedings of the USENIX Security Symposium* (2018).

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

[13] HONNIBAL, M., AND MONTANI, I. spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks, and Incremental Parsing. *To appear* (2017).

[14] JOHNSON, S. C. Lint, a C Program Checker. In *COMP. SCI. TECH. REP* (1978), pp. 78–1273.

[15] LIN, J., SADEH, N., AND HONG, J. I. 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).

[16] LIU, F., RAMANATH, R., SADEH, N., AND SMITH, N. A. A Step Towards Usable Privacy Policy: Automatic Alignment of Privacy Statements. In *Proceedings of the International Conference on Computational Linguistics (COLING)* (2014).

[17] MARELLA, A., PAN, C., HU, Z., SCHAUB, F., UR, B., AND CRANOR, L. F. Assessing Privacy Awareness from Browser Plugins.

[18] MCDONALD, A. M., AND CRANOR, L. F. The Cost of Reading Privacy Policies. *I/S Journal of Law and Policy for the Information Society (ISJLP) 4* (2008).

[19] NORTON, T. B. Crowdsourcing Privacy Policy Interpretation. *Proceedings of the Research Conference on Communications, Information, and Internet Policy (TPRC)* (2015).

[20] RAMANATH, R., SCHAUB, F., WILSON, S., LIU, F., SADEH, N., AND SMITH, N. A. Identifying Relevant Text Fragments to Help Crowdsource Privacy Policy Annotations. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing (HCOMP)* (2014).

[21] RAO, A., SCHAUB, F., SADEH, N., ACQUISTI, A., AND KANG, R. Expecting the Unexpected: Understanding Mismatched Privacy Expectations Online. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)* (2016).

[22] SLAVIN, R., WANG, X., HOSSEINI, M. B., HESTER, J., KRISHNAN, R., BHATIA, J., BREAUX, T. D., AND NIU, J. Toward a Framework for Detecting Privacy Policy Violations in Android Application Code. In *Proceedings of the International Conference on Software Engineering (ICSE)* (2016).

[23] STAMEY, J. W., AND ROSSI, R. A. Automatically Identifying Relations in Privacy Policies. In *Proceedings of the ACM International Conference on Design of Communication (SIGDOC)* (2009).

[24] WANG, X., QIN, X., HOSSEINI, M. B., SLAVIN, R., BREAUX, T. D., AND NIU, J. GUILeak: Tracing Privacy Policy Claims on User Input Data for Android Applications. In *Proceedings of the International Conference of Software Engineering (ICSE)* (2018).

[25] YU, L., LUO, X., LIU, X., AND ZHANG, T. Can We Trust the Privacy Policies of Android Apps? In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2016).

[26] ZAEEM, R. N., GERMAN, R. L., AND BARBER, K. S. PrivacyCheck: Automatic Summarization of Privacy Policies Using Data Mining. *ACM Transactions on Internet Technology (TOIT)* (2013).

[27] ZIMMECK, S., AND BELLOVIN, S. M. Privee: An Architecture for Automatically Analyzing Web Privacy Policies. In *Proceedings of the USENIX Security Symposium* (2014).

[28] ZIMMECK, S., WANG, Z., ZOU, L., IYENGAR, R., LIU, B., SCHAUB, F., WILSON, S., SADEH, N., BELLOVIN, S. M., AND REIDENBERG, J. 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 to users via a link within the Google Play Store. The link leads to a page on the developer's website containing the policy in HTML format. Most NLP parsers expect plaintext input, therefore PolicyLint begins by converting the HTML privacy policy into plaintext. Conversion from HTML is challenging for several reasons. First, HTML policies may contain non-displayable elements or non-relevant text to the privacy policy, such as navigation links. Second, HTML documents occasionally contain pop-up elements, which may interrupt the flow of text. Third, NLP parsers have difficulty consuming formatted lists, resulting in problems such as detecting incorrect sentence breaks or mistagging parts-of-speech and typed dependencies. We now describe how PolicyLint overcomes these challenges.

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 the policies that are 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 of the HTML documents, PolicyLint must ensure the plaintext document has a formatting style similar to the text displayed on the webpage (e.g., the 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, an 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 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 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 structure and nesting depth of items. 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 follows Algorithm 1 to form complete sentences by merging the list item text with the preceding text.

Algorithm 1 proceeds as follows. PolicyLint iterates over the paragraphs in the markdown document to find paragraphs that end in a colon. For each paragraph that ends in a colon, PolicyLint checks whether the proceeding paragraph is a list item (e.g., starts with a common bullet points, roman numerals, numbers, or was annotated during the HTML processing stage). If the line of text is a list item, PolicyLint creates a new paragraph by appending the list item text to the preceeding text that ended with the colon. If the list item ends in another colon, PolicyLint repeats the same process above but by prepending the nested list items to the new paragraph created in the last step. PolicyLint then leverages the symbols

Algorithm 1 Merging Formatted Lists

```
1: procedure MERGELIST(docPos, currentLine, prependTxt, itemSymb)
2:     if endsWithColon(currentLine) then
3:         prependTxt ← prependTxt + currentLine
4:         while nxtLine ← getNextLineInDoc(docPos++) do
5:             if beginsWithItemSymbol(nxtLine, itemSymbol) then
6:                 itemSymbol ← predictNextItemSymbol(nxtLine)
7:                 newTxt ← prependTxt + stripSymbols(nxtLine)
8:                 if endsWithColon(nxtLine) then
9:                     docPos ← MergeList(docPos, nxtLine, newTxt, NULL)
10:                else
11:                    write(newTxt)
12:            else
13:                return docPos − 1
14:    return docPos
```

that denote list items (e.g., bulletpoints, numbers, letters, roman numbers) to predict the next item in the list's expected symbol, which is useful for detecting boundaries of nested lists. For example, if the current list item is started with "(1)," then we would expect the next list item to be started with "(2)." If the item symbol matches to 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 the markdown to plaintext. During this process, PolicyLint strips markdown formatting such as header tags and bullet points, normalizes unicode characters, and strips list item numbering and other format characters. Finally, PolicyLint uses `langid` to determine if 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

As discussed in Section 4, PolicyLint requires a training set of sharing and collection sentences to learn underlying patterns from in order to identify "unseen" sharing and collection sentences. As manually selecting a set of sharing and collection sentences with diverse grammattical structures is a tedious process, we opted to auto-generate the training sentences for PolicyLint 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 the new sentence. To identify the templates, we used our domain expertise to identify different sentence compositions that could describe sharing and collection sentences. We identified 16 sentence templates, as shown in Table 4.

To fill the templates, we need to 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 began 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 identified 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."

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 included conjuncts of the *DATA* and *ENT* placeholders to account for deviations in the parse tree due to syntactical ambiguity (i.e., a sentence can have multiple interpretations). For example, consider the sentence, "We share your personal information with advertisers". In one interpretation, the prepositional phrase "with advertisers" modifies the verbal phrase "share your personal information." However, a second interpretation could be that "with advertisers" modifies the noun phrase "your personal information." We found that the NLP parser [13] that we used produced a parse tree with first interpretation when the placeholder was filled with a singular value, but produces a parse tree with the second interpretation when the placeholder was filled with a conjunct of data objects and or entities. Therefore, we generate two sentences for each template: one with a singular *DATA* and *ENT*, and second with the plural *DATA* and plural *ENT*.

To fill the templates, we iterate through each SoC verb and each template and fill in 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 templates for "send", "give", and "provide" if the template does not contain a placeholder for a preposition (i.e., T1, T6, T8, T10, T12, T14), as those sentences did not make sense sence without specifying to *whom* the data is being sent/given/provided. We generate two sentences for each template when filling in the *DATA* and *ENT* placeholders to try to account for syntactic ambiguity. The first sentence is with the singular cases of *DATA* and *ENT* and the second sentence is with the conjuncts of *DATA* and *ENT*. For example, consider the first template and the verb "share." In this case, we would produce the sentences, "We may share your personal information with advertisers.", and "We may share your personal information, demographic information, and financial information with advertisers, analytics providers, and our business partners." In total, we generate 560 sentences, which are used by PolicyLint to learn patterns from to identify sharing and collection sentences.

C NER Performance

D Term Lists

Table 4: 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.

Table 5: NER Performance: Comparison of spaCy's stock en_core_web_lg model versus our domain adapted 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%

Table 6: 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

Table 7: 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

Table 8: 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