Toward a Framework for Detecting Privacy Policy Violations in Android Application Code

Rocky Slavin[1], Xiaoyin Wang[1], Mitra Bokaei Hosseini[1], James Hester[2], Ram Krishnan[1], Jaspreet Bhatia[3], Travis D. Breaux[3], and Jianwei Niu[1]
[1]University of Texas at San Antonio, San Antonio, TX, USA
[2]University of Texas at Dallas, Dallas, TX, USA
[3]Carnegie Mellon University, Pittsburgh, PA, USA
{rocky.slavin, xiaoyin.wang, mitra.bokaeihosseini, ram.krishnan, jianwei.niu}@utsa.edu
william.hester@utdallas.edu, {jbhatia, breaux}@cs.cmu.edu

ABSTRACT

Mobile applications frequently access sensitive personal information to meet user or business requirements. Because such information is sensitive in general, regulators increasingly require mobile-app developers to publish privacy policies that describe what information is collected. Furthermore, regulators have fined companies when these policies are inconsistent with the actual data practices of mobile apps. To help mobile-app developers check their privacy policies against their apps' code for consistency, we propose a semi-automated framework that consists of a policy terminology-API method map that links policy phrases to API methods that produce sensitive information, and information flow analysis to detect misalignments. We present an implementation of our framework based on a privacy-policy-phrase ontology and a collection of mappings from API methods to policy phrases. Our empirical evaluation on 477 top Android apps discovered 341 potential privacy policy violations.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Documentation

Keywords

Privacy Policies, Android Applications, Violation Detection

1. INTRODUCTION

In early 2015, the Android operating system (Android) accounted for 78.0% of the worldwide smartphone market share [5]. With this sizable market share comes an increase to end user privacy risk as mobile applications (apps) built for the Android have access to sensitive personal information about users' locations, network information, and unique device information. To protect privacy, regulators, such as the U.S. Federal Trade Commission (FTC), have relied on natural language privacy policies to enumerate how applications collect, use, and share personal information. Recently, the California Attorney General Kamela Harris negotiated with the Google Play app store to require mobile app developers to post privacy policies [21]. Despite this effort to produce these policies, as with any software documentation, there are opportunities for these policies to become inconsistent with the code. These policies can be written by people other than the developers, such as lawyers, or the code can change while the policy remains static. Such inconsistencies regarding an end user's personal data, intentional or not, can have legal repercussions that can be avoided with proper consistency checks. For example, the FTC, under their unfair and deceptive trade practices authority, requires companies to be honest about their data practices in their privacy policies. Companies, such as SnapChat, Fandango, and Credit Karma, often settle with the FTC for inconsistent policies and practices by accepting 20 years of costly privacy and security audits [1, 2]. It is therefore good practice for mobile apps to clearly state in their privacy policies what data is collected and for what purpose. For large companies, this task is commonly assigned to a team of legal experts, however, mobile app developers are frequently small start-ups with 1-5 developers [19] where such a task is not easily assigned.

It is important for software engineers to be aware of the data their code is collecting along with what their policy says they are collecting not only for legal reasons, but for the production of quality apps. As more data is entrusted to technology, end users become more aware of the ramifications of mishandled private data [27]. Thus, software engineers are entrusted by end users to not only care for their data, but disclose what exactly is being collected.

In this paper, we present three **contributions**: **(1)** an empirically constructed mapping from policy phrases to private-data-producing Android Application Program Interface (API) methods that has been compiled from real-world app policies and API documentation. The many-to-many map links 76 commonly used data collection phrases and their synonyms to 154 Android API method signatures. **(2)** We created an approach that identifies privacy promises in mobile app privacy policies and checks these against code using information flow analysis to raise potential policy violations. As part of checking for data over-collection violations within the app, the approach uses information flow analysis to see if the data is sent outside the app. **(3)** We constructed an initial ontology of 368 data collection phrases to use in conjunction with the API mappings. The ontology provides a means to increase the phrase coverage of the mappings without the need for analysis on more apps and privacy policies.

ICSE' 16, May 14-22, 2016, Austin, Texas, USA

This paper is organized as follows: in Section 2, we review the background upon which we based our approach; in Section 3, we describe the manual process used in the creation the framework; Section 4 describes our automated method for privacy policy violation detection; Section 5 describes the evaluation of our approach followed by discussion of the results and approach in Section 6; Section 7 includes related work; Section 8 describes our plans for future work and we conclude in Section 9.

2. BACKGROUND

This section presents the background upon which our research is based.

2.1 Android Operating System

Android is an open source mobile operating system (OS) based on more than 100 open source projects including the Linux kernel. Android is developed by Google and has been reported more popular as a target platform for developers than iOS in 2015 which makes it the most popular mobile operating system today [3]. Apps made to run on Android can be downloaded from multiple repositories, the most popular being Google Play[1]. In 2014, Google revealed that there were more than one billion active monthly Android users [38]. These characteristics make it attractive to startups and established companies alike.

Android utilizes the Linux security model and layers through a user-based permission system [42]. Apps can access resources through the permission system to gain access to resources such as the camera, GPS, Bluetooth, telephony functions, network connections, and other sensors [42]. Such permissions are granted to apps by users when they install an app. All permissions not listed to and subsequently granted by the user are denied to the app [41]. Although Android applies this permission system and rigorous security management, data leakage and misuse is still possible [15]. This can be due to problems with the current Android permission system such as low granularity of the permissions [26] and the ambiguity of the phrases presented to users when installing an app [23]. Problems such as these can allow apps to access the sensitive data by calling Android Application Program Interface (API) methods in the app source code.

2.2 Application Program Interface

Applications can interact with underlying Android system using a framework API provided by the Android platform. The framework API contains set of packages, classes, and methods. The Android 4.2 framework is comprised of about 110,000 methods, some of which are specifically used to retrieve, insert, update, or delete sensor data through the Android OS [29]. The use of an API increases the level of security by not allowing apps to have direct access to all sensor data by default.

Before an app can access specific methods from the API, the required permissions must be requested by the app through a manifest file. An app's manifest file enumerates the app's required permissions and is described to users when installing an app as well as on the app's download page on the Google Play store. Thus, there is a direct relationship between the permissions granted to an application by a user at installation time and eligible API method calls in the application source code.

2.3 Privacy Policy

Besides the standard permissions for API access documented in manifest files, applications' privacy policies are a source for identifying what information is collected and used by apps.

A privacy policy serves as the primary means to communicate with users regarding which and how sensitive personal information (SPI) has been accessed, collected, stored, shared (app to app, and to third party), used/processed, and the purpose of the SPI collection and processing. Privacy policies generally consist of multiple paragraphs of natural language such as the following excerpt from the Indeed Job Search app's privacy policy[2] listed on Google Play:

> Indeed may create and assign to your device an identifier that is similar to an account number.We may collect the name you have associated with your device, device type, telephone number, country, and any other information you choose to provide, such as user name, geo-location or e-mail address. We may also access your contacts to enable you to invite friends to join you in the Website.

Privacy policies are particularly important in the United States due to the "notice and choice" approach used to address privacy online [31]. Under this framework, app companies post their privacy policies and users read the policies to make informed decisions on accepting the privacy terms before installing the apps [31]. However, most privacy policies prepared by policy authors are difficult to understand due to their verbose and ambiguous nature, and this can lead to users to skip reading policies even if they have concerns about information collection practices. More significantly, the app developers might not be able to comply with privacy policies effectively. To address this issue, this work aims to provide a framework to achieve alignment between apps' privacy policies and implementation code, and better communication among software developers and policy writers.

A major hindrance in the understanding and analysis of privacy policies is that there is no canonical format for presenting the information. The language, organization, and detail of policies can vary from app to app.

3. MANUAL PREPARATION

The goal of this work is to discover information regarding the relationship between terminology used in privacy policies expressed in natural language and API method calls used in the corresponding code. Such a mapping would then provide semantic information regarding the natural language. In turn, an app's source code could more easily be checked for misalignment with its corresponding privacy policy. Before we can perform such an automated detection of privacy policy violations, we must construct initial data sets and a mapping from which the knowledge can be used to detect violations in other apps. The following subsections describe how we leveraged a small subset of Android apps' source code to implement a mapping from API methods to policy phrases. This information is is then used to detect violations in a much larger set of Android apps (discussed in Section 5).

In our approach, we created a mapping between API method signatures in the Android SDK and meanings shared between API documents and privacy policies. The shared meanings are described in an ontology that provides support for comparing two technical terms: we say that one term subsumes a second term, when either the first term is more general than the second term, called a *hypernym*, or when the second term is part of the first term, called a *meronym*. For example, "mobile device model" and

[1]https://play.google.com/

[2]http://www.indeed.com/legal

"sensors" are parts of a "mobile device," whereas "mobile device model" is also a kind of "mobile device information." In addition, we define two terms as *synonyms* when the meaning is equivalent for our purposes (e.g., when "IP address" is a synonym for "Internet protocol address"). Because privacy policies tend describe technical information using more generic concepts, the ontology allows us to map from low-level technical terms to high-level technical categories, and vice versa. Once the ontology is constructed, we can use tools to automatically infer which terms should appear in privacy policies based on the API method calls in a mobile application.

We now describe how we created the ontology and mapping by extracting terminology from the privacy policies and API documents respectively, before we classified this terminology using subsumption and equivalence relationships. In each step, we employed research methods aimed at improving construct and internal validity and reliability, which we discuss.

3.1 Extracting the API Terminology

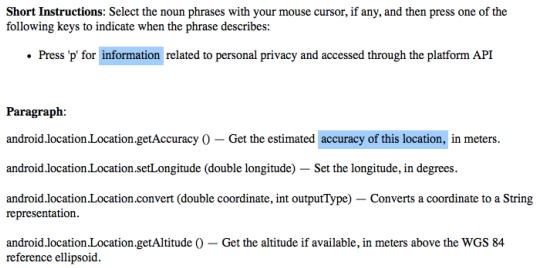
In our approach, a subject matter expert, who would typically be the maintainer of the Application Programmer Interface (API) documentation, annotates an API document. The annotations map key phrases in the API documents to low-level technical terminology in an API lexicon (e.g., "scroll bar width" or "directional bearing" are low-level technical terms). To bootstrap our approach, we chose to annotate the entire collection of API documents in the Android SDK, which includes 2,988 API documents containing over 6,000 public method signatures (here, the term "public" refers to the Java access modifier). Each API document consists of one or more method signatures, which each consist of the method name, input parameters, the return type, and a natural language description of the method's behavior.

The annotation procedure involves three steps: (a) we extract the method names, input parameters and natural language method descriptions from the API documentation to populate a series of crowd worker tasks; (b) for each crowd worker task, two investigators separately annotate the extracted fields by identifying which phrases correspond to a kind of privacy-related platform information; and (c) the resulting annotations are compiled into a mapping from the fully qualified method name, including API package name, onto each annotated phrase (i.e., each method name can map to one or more platform information phrases). We only compiled mappings where the two investigators both agreed that the phrase was a kind of privacy-related, platform information.

In the first step, the signatures were automatically extracted from the API documents, which were themselves expressed in HTML generated using the Javadoc toolset. The signatures were then segmented into sets of 20 signatures or less, and each set was presented in a separate crowd worker task. Applying the segmentation to the 2,988 API documents yields 310 crowd worker tasks.

The crowd worker task employs a web-based coding toolset developed by Breaux and Schaub [11] for annotating text documents using coding theory, a qualitative research method for extracting data from text documents [33]. In coding theory, the annotators use a coding frame to decide when to code or not to code a specific item. In our study to annotate the API documents, our coding frame consisted of a single information code defined as information "related to personal privacy and accessed through the platform API." In the second step, two investigators used this web-based toolset to code the 310 crowd worker tasks, consuming 6.5 and 6.6 hours for each investigator to yield 195 and 196 annotations, respectively.

Figure 1 shows an excerpt from the crowd worker task, where a worker has annotated phrases in the Location package of the Android API. The toolset has been validated in a prior case study to extract privacy requirements from privacy policies [11]. The toolset also includes analytics for extracting overlapping annotations where n or more workers agreed that the phrase should be annotated.


Short Instructions: Select the noun phrases with your mouse cursor, if any, and then press one of the following keys to indicate when the phrase describes:

- Press 'p' for information related to personal privacy and accessed through the platform API

Paragraph:

android.location.Location.getAccuracy () — Get the estimated accuracy of this location, in meters.

android.location.Location.setLongitude (double longitude) — Set the longitude, in degrees.

android.location.Location.convert (double coordinate, int outputType) — Converts a coordinate to a String representation.

android.location.Location.getAltitude () — Get the altitude if available, in meters above the WGS 84 reference ellipsoid.

Figure 1: API Annotation Crowd Worker Interface

From the two investigator's combined annotations, we produced 219 unique annotations with duplicate annotations removed. The total 219 annotations were next compiled into a mapping between API method signatures and annotated phrases. The phrases in the mapping were normalized by the two investigators by converting the annotated text into simple noun phrases (described further in Section 3.4). This is necessary to reduce the variety of ways that method behaviors are described into a concise, reusable API lexicon. The resulting lexicon contains 162 unique phrases and 169 total mappings between phrases and API method names. A total of 154 methods were annotated based on the criteria that they produce privacy related information.

3.2 Extracting the Privacy Policy Terminology

Each app page on Google Play includes a link to the app's privacy policy if it is specified by the developer. We created a Python script to download the privacy policies from these links for the top 20 free apps in each app category[3]. We filtered theses policies based on their formatting, language (we only considered policies written in English), and whether or not a "Privacy Policy" section was explicitly stated in the document and randomly selected 50 from this pool for terminology extraction.

For our approach, we determine which kinds of technical information should appear in privacy policies to describe privacy-relevant API method calls. To bootstrap our method, we developed a privacy policy lexicon in which six investigators annotated the 50 mobile app privacy policies using our crowd worker task toolset [11]. Unlike the API lexicon, wherein we used only two investigators with programming experience, we used six annotators for extracting terms from privacy policies, because privacy policy terminology includes vague and ambiguous terms that span a broader range of expertise (e.g., "taps" corresponds to user input, whereas "analytics information" includes web pages visited, links clicked, browser information, and so on.) Thus, by increasing the number of annotators, we increased our likely coverage of potentially relevant policy terms.

The crowd worker task employs the same web-based coding toolset developed by Breaux and Schaub [11]. To prepare the policies for annotation, we first removed the following content: the introduction and table of contents, "contact us", security, U.S. Safe Harbor, policy changes and California citizen rights. This content generally appears in separate sections or paragraphs, which

[3]The list of app categories is available at the Google Play website, and the top 20 apps for each category was fetched on May 19th 2015.

reduces the chance of inconsistency when removing these sections across multiple policies. While these sections do describe privacy-protecting practices, such as complying with the U.S. Safe Harbor, we have never observed descriptions of platform information in our analysis of over 100 privacy policies in our previous research [12]. Next, we manually split the remaining policy into spans of approximately 120 words. We preserve larger spans which either have an anaphoric reference back to a previous sentence (e.g. when "this information..." depends on a previous statement to understand the context of the information), or when the statement has subparts (e.g., (a), (b) etc.) that depend on the context provided by earlier sentence fragments. On average, we need 15 minutes per policy to complete the preparation.

The coding frame for the privacy policy terminology extraction consists of two codes: *platform information*, which we define as "any information that $company or another party accesses through the mobile platform, which is not unique to the app;" and *other information*, which we define as "any information that $company or another party collects, uses, shares or retains." We replace the $company variable with the name of the company whose policy is being annotated. Next, we compiled the annotations where two or more investigators agreed that the annotation was a kind of platform information; we excluded non-platform information from this data set. We applied an entity extractor [9] to the annotations to itemize the platform information types into unique entities, which were then included in the privacy policy lexicon.

Among the 50 policies, we constructed 5,932 crowd worker tasks with an average word count of 98.6; the average words per policy was 2054.6. These tasks produced a total of 720 annotations across the 50 policies, which yielded a total of 368 unique platform information entities. The total time required to collect these annotations was 19.9 hours across six annotators, all of whom are authors of this paper. We now discuss how we created a platform information ontology from this lexicon.

3.3 Constructing the Ontology

A common phenomena in natural language description is generalization, in which a more general phrase can be used to imply a number of sub-concepts of the phrase. For example, the phrase "technical information" may imply a wide range of technical data, while the phrase "device identifier" is more specific, but its concept is still covered by phrase "technical information". Since phrase generalization is often used to describe information collected, it is important to be able to distinguish these relationships between phrases in order to identify cases where a concept is represented in another phrase. To handle this, we created an ontology of privacy-related phrases to be used as a cross reference during the identification of methods not represented in privacy policies.

An ontology is a formal description of entities and their properties, relationships, and behaviors [20], and is described with formal languages such as OWL (based on Description Logic). In the context of phrase mapping, we use an ontology to represent a hierarchical classification of phrases. For example, in Figure 2, "IP Address" is a decedent of "Network Information", indicating that IP Address is a type of network information. The hierarchical nature of an ontology allows for transitive relationships that can be used for mapping API methods to phrases indirectly based on relationships between the phrases themselves.

The ontology is used to formally reason about the meaning of terminology found in the API documents and privacy policies. For an API lexicon $\hat{A}$ and a privacy policy lexicon $\hat{P}$ consisting of unique terms (or concepts), the ontology is a Description Logic (DL) knowledge base KB that consists of axioms $C \sqsubseteq D$, which

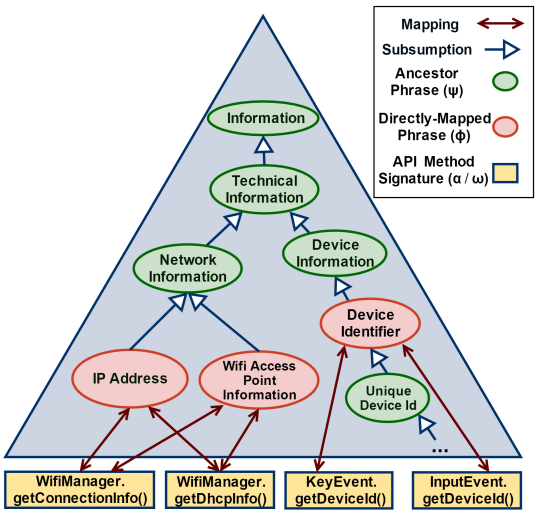


Figure 2: Abbreviated Ontology Example with Mapped API Methods

means concept C is subsumed by concept D, or $C \equiv D$, which means concept C is equivalent to concept D, for some concepts $C, D \in (A \cup P)$. Using our API lexicon, our aim is to map a method name m from an API document to a concept $A \in \hat{A}$. Next, we aim to infer (in a forward direction) all policy concepts $\{P | P \in \hat{P} \wedge KB \models P \sqsubseteq A \vee KB \models P \equiv A\}$. In this respect, we can extract method names from method calls in a mobile app, then infer corresponding policy terms (among which at least one) should appear in the mobile app's privacy policy. Similarly, we can reason in the backward direction to check which policy terms mentioned in the app's policy map to which method names corresponding to method calls in the app.

We constructed the ontology following a method developed by Wadkar and Breaux [37]. First, we generated a basic ontology consisting of one concept for each term in the privacy policy lexicon; each concept was subsumed by the $\top$ concept, and no other relationships among concepts existed. Second, for two copies of the basic ontology KB_1 and KB_2, two investigators separately performed pairwise comparisons among term pairs C, D in each ontology, respectively: if two terms were near synonyms, the first investigator created an equivalence relation $KB_1 \models C \equiv D$; else, if one term subsumed the other term, the first investigator created a subsumption relationship $KB_1 \models C \sqsubseteq D$. Due to the number of pairwise comparisons, it's not unreasonable to expect that a single investigator would produce an incomplete ontology, or an ontology that is inconsistent with another investigator's ontology. To check for completeness and consistency between two investigators, we compared all relationship pairs between KB_1 and KB_2, including cases were a relationship did not exist in one of the ontologies. Both investigators met to reconcile any differences, recognizing that classification differences can persist forward into our analysis of mobile app violations.

For two investigators, the resulting ontologies KB_1 and KB_2 consisted of 431 and 407 axioms, respectively. The first comparison yielded 321 differences and was evaluated using Cohen's Kappa to measure the degree of agreement above chance alone [14], which was 0.233. After the reconciliation process, the investigators were left with 12 differences and a Cohen's Kappa of 0.979.

3.4 Constructing the Mapping

With the ontology constructed from the privacy policy lexicon,

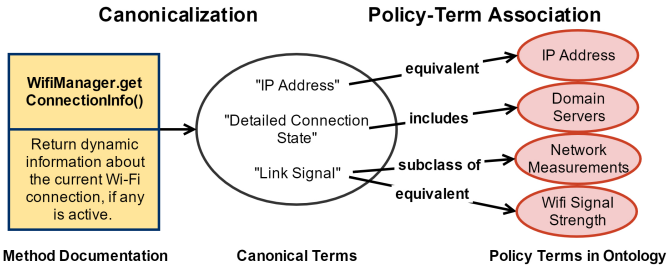


Figure 3: Mapping Process

individual API methods could then be mapped to one or more terms in the ontology based on their annotations from the API lexicon as well as their return types. Figure 3 shows how intermediate noun phrases were created as a canonical representation of the method's description and then mapped directly to terms in the ontology based on their relationships. This canonicalization process made explicit the domain knowledge about the methods (i.e., canonical terms) and the natural language used to describe the method in privacy policies (i.e., terms in the ontology). As exemplified in the figure, the documentation describes "dynamic information about the current Wi-Fi connection" as the data it produces. In cases such as this, where the description did not explicitly describe the information returned, we analyzed the object returned by the method. Here, the object (of type *WifiInfo*) provided multiple public fields and methods from which we were able to assign the canonical terms in the figure (as seen in the white circle). From there, the canonical terms were associated with related terms in the ontology based on their relationships. This effectively produces a mapping between each of the API methods and one or more terms in the ontology (assuming the method is privacy-relate) and vice versa. We refer to this many-to-many mapping relation, of which each element is a pair, $(policy\ term, API\ method)$, as $Mappings$ in the following sections.

4. AUTOMATED VIOLATION DETECTION

To detect potential privacy policy violations, we first identify API method invocations that produce data covered by a known policy term from the privacy policy lexicon. Next, we use information flow analysis to check whether that data flows to a remote server via a subsequent network API method invocation. Data collected by a method is considered a potential privacy policy violation if the method is not represented in the app's privacy policy through $Mappings$. An overview of the full process is in Figure 4.

4.1 Weak and Strong Violations

As discussed in Section 2.3, privacy policies serve to inform users about how their personal information is collected and used. These policies cover a wide range of practices, including in-store, client-side, and server-side practices, and they may describe all of a company's practices, or be limited to only those practices of a single product or service. In this paper, we are only concerned about client-side practices affecting mobile applications. In addition, privacy policies are not complete: they generally describe a subset of the company's practices. Therefore in our approach, we only detect *errors of omission*, in which the app collects a kind of information that is not described in the policy. Errors of omission are *potential* policy violations, because the collection may be unintended by the app developer. Moreover, because privacy includes notifying users about how their information is collected and used, errors of omission represent potential privacy violations. We detect two kinds of violations resulting from errors of omission: *strong violations* that occur when the policy does not describe an app's data collection practice, and *weak violations* that occur when the policy describes the data practice using vague terminology. Other kinds of policy errors, such as *direct conflicts*, in which a conflict occurs because the policy states that an app does not collect a kind of information and the app does indeed collect that kind of information, are out of scope of this paper.

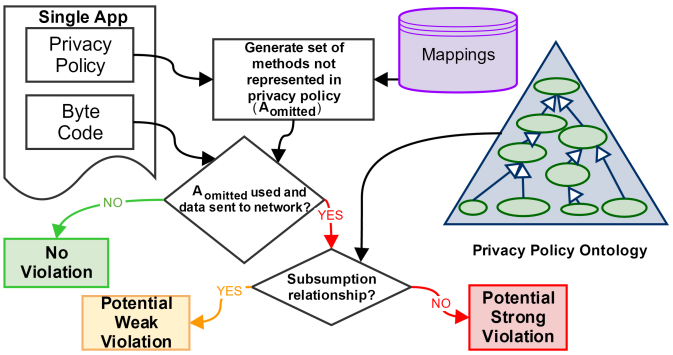


Figure 4: Violation Detection

4.2 Detection of Suspicious Method Invocations

To begin the process, we preprocess the app's privacy policy to generate a set of method-related phrases that represent what the privacy policy states it collects. First, all words in each policy paragraph are converted to their base dictionary form (i.e., lemmatization) [25]. If the lemmatized paragraph contains a collection verb[4], the paragraph is kept for further analysis. Next, the intersection of the lemmatized words in the paragraphs and the phrases in the privacy policy ontology is calculated to produce the set of existing policy phrases Φ. We use Φ and the many-to-many relation $Mappings$ to generate a list of method names, $A_{represented}$,

$$A_{represented} = \{\alpha \mid \phi \in \Phi \wedge \alpha \in map(\phi)\} \quad (1)$$

where $A_{represented}$ denotes the set of methods that are directly represented within the privacy policy based on the mapping, and $map(\phi)$ produces all the methods to which ϕ is mapped.

The set of omitted API method names can then be defined as the following, where A_{mapped} represents the set of API method names that appear in $Mappings$ (i.e., all methods in the Android API for which at least one mapping to a policy phrase exists).

$$A_{omitted} = A_{mapped} \setminus A_{represented} \quad (2)$$

The app's source code can now be scanned for any instances of $\alpha \in A_{omitted}$. Such instances would then be flagged as a suspicious invocation, ω. To determine information about the invocation, the offending API method name, ω, is cross-referenced with A_{mapped} to determine all phrases to which it is mapped, Φ_ω. The ontology is then used to determine the set of terminological ancestors, Ψ_ω, of all $\phi \in \Phi_\omega$ (Equation 4), which are those phrases that include in their interpretation the data accessed by the API method invocation. The phrases Ψ_ω semantically subsume (i.e., are more generally descriptive of) at least one member of Φ_ω according to the ontology. The relationships

[4]The collection verbs used in this study are the result of the manual annotation by two investigators of 25 random privacy polices from the set of 50 privacy policies described in Section 3.2.

between α, ω, Φ, and Ψ can be seen in Figure 2, where an α becomes an ω if it is a suspicious invocation. Due to the nature of an ontology, the set of nodes of highest level $HNodes = \{\text{“}information\text{”}, \text{“}software\text{”}, \text{“}technology\text{”}\}$ is generally descriptive of all of their subterms and thus not useful.

Therefore, we use Equation 3 to describe the ontology.

$$O' = O \setminus HNodes \tag{3}$$

$$\Psi_\omega = \{\psi \mid \phi \in \Phi_\omega \wedge \phi \sqsubseteq \psi\} \tag{4}$$

The members of Ψ_ω represent terminological ancestors of ω that relate to ω through a subsumption relationship. These ancestors may include other interpretations that are not associated with ω (e.g., “technical information” includes “location information” and “usage information”, which are distinct and different kinds of information). Thus, we check $\psi \in \Psi_\omega$ for matches in the privacy policy to determine if ω is described in the policy through a more general term. If a match is found, then a resulting violation is flagged as a weak violation (i.e., the policy contains a phrase transitively mapped to an API method name). Otherwise, the violation is flagged as a strong violation, which means there is no relationship between any phrase in the policy, the ontology, and the corresponding API method invocation. While a strong violation is obviously harmful to the protection of the users' privacy due to the lack of notice, a weak violation is still potentially harmful, because it indicates the lack in sufficient detail about the data practice and it can be used as a guide for improving the clarity of the privacy policy [30].

4.3 Information Flow Analysis

As explained above, an API method invocation becomes a violation only if the information it fetches is sent to remote servers. Therefore, we need to further check the destination of the fetched data, and existing information flow analysis tools provide the technique needed for this goal. We also require a list of *sink* methods that send information to remote servers.

It should be noted that our framework works with any information analysis tool and sink methods list for network data transfer. In our implementation, we leveraged FlowDroid [6], the state-of-art technique for Android information flow analysis, to track the information flow within Android byte code. We also used the list of sink methods for network data transfer described by SUSI [29], a machine-learning tool for classifying Android sources and sinks.

5. EMPIRICAL EVALUATION

In this section, we present an empirical evaluation of our framework by applying it to top Android apps and their privacy policies. In the evaluation, we try to answer the following research questions.

- **RQ1:** Is our framework able to detect violations of privacy policies in real-world Android apps?
- **RQ2:** How do the techniques in our framework affect its effectiveness on violation detection?
- **RQ3:** What are the major types of privacy information that are silently collected in detected privacy policy violations?

5.1 Study Setup

In this subsection, we introduce how we construct the data set for empirical evaluation, the metrics used, and the compared variants of our framework.

5.1.1 Data Collection

The first step in our evaluation is to construct a data set of Android apps with their corresponding privacy policies. In particular, from the official Google Play market, we downloaded the top 300 free apps[5], as well as the top 20 free apps for each app category[6]. We combined all the downloaded apps and acquired an app data set of 1,096 apps. Note that, although most Android apps have privacy policies, the app owners may put the policy at different places, such as their portal site at Google Play market, or a link in the main page of their company/organization. The privacy policy can also be in different formats, such as HTML, PDF, or Windows Word Document. Based on our observation, a large proportion of apps place their corresponding privacy policy at their portal websites at the Google Play market. Therefore, we crawled these websites and tried to automatically download the privacy policies of these apps. Furthermore, we considered only HTML privacy policies (the most popular format of privacy policies) in our evaluation for simplicity and avoiding potential noise in text extraction from various file formats. Note that, with proper text extraction tools, our framework can be applied to any format of privacy policies. Based on the automatic downloading and file format filtering, we collected privacy policies for 477 of the 1,096 apps, and thus generated a data set with 477 apps and their corresponding privacy policies[7].

5.1.2 Evaluation Metrics

To answer research questions **RQ1** and **RQ2**, we needed to measure the effectiveness of violation detection. In our study, we measured the effectiveness of our framework by the number of violations detected, the number of true positives, false positives, and violations whose types (strong violations or weak violations) are mis-identified. It should be noted that, the ground-truth number of true violations in the entire data set is unknown, and thus the number of false negatives can not be calculated. In particular, we determined whether a detected violation is a true violation by manual inspection, and each violation was assigned to and inspected by two of the authors. For disagreements, a third author was assigned for reconciliation. When counting violations, we considered all invocations of the same API in an app as one violation.

5.1.3 Evaluated Techniques

To answer research question **RQ2**, we considered two variants of our framework, and compared them with our default technique.

In our API mapping, we leveraged the knowledge from Android official documentation and crowd sourcing techniques to generate a fine-grained mapping between API methods and phrases in privacy policies. As a means to evaluate this fine-grained mapping, the first evaluation variant (referred to as “SUSI-only”) used the coarse-grained SUSI API categorizations to map API methods to phrases. Specifically, we assigned phrases in the ontology to their most relevant SUSI categories (e.g., all descendants of the phrase “unique identifiers” in the ontology are assigned to the category of “unique identifiers”). Note that, since we focus on the collection of platform information only, the API methods in our mapping fall into one of the following 5 categories in SUSI: Unique Identifiers, Location Information, Network Information, Bluetooth Information, and No-Category (a SUSI category for API methods that are

[5]The ranking of top 300 free apps is available at the Google Play website and was fetched on May 19th 2015.

[6]The list of app categories is available at the Google Play website, and the top 20 apps for each category were fetched on May 19th 2015.

[7]The data set is available at our project website: http://sefm.cs.utsa.edu/android_policy.

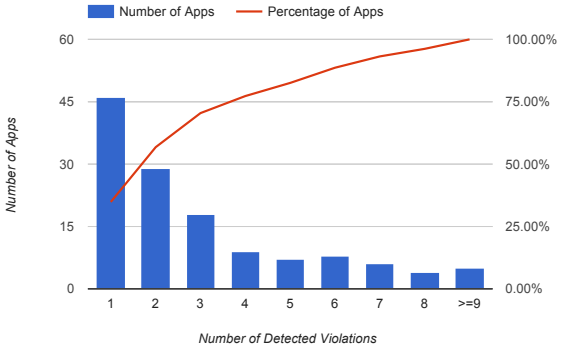


Figure 5: Distribution of Apps on the Number of Detected Violations

difficult to categorize), so we also assigned our policy phrases to these 5 categories. After the assignment of phrases, we mapped all phrases in a category in SUSI categories to all the API methods in the same category, and thus generated a new coarse-grained mapping based on SUSI only.

A second variant (referred to as "Keyword Search") was used to study the effectiveness of using light-weight NLP techniques, such as lemmatization, to filter out irrelevant paragraphs in privacy policies (described in Section 4.2). Specifically, this method did not use any filtering techniques and instead used a simple keyword-search-based strategy to extract phrases, Φ, from the privacy policies.

5.2 Study Results

Overall Results. We applied our default technique to the 477 pairs of apps and privacy policies in our data set. For the 477 apps, with a 30 minute time-out limit for each app, FlowDroid successfully processed 375 of them[8]. From these 375 apps, our default technique detected 402 violations in total, including 58 strong violations and 344 weak violations. Our manual inspection revealed that, among these detected violations, 341 were true violations, including 74 strong violations and 267 weak violations (note that our framework mistakenly classified 19 strong violations as weak violations). The detection of 267 true weak violations shows that our privacy-phrase ontology is helpful on differentiate weak violations from strong violations.

We further studied how these violations were distributed among the apps, and the result is shown in the Pareto chart in Figure 5. In the figure, the blue bars represent the number of apps that have a certain number of violations, and the red line represents the proportion of apps that have violations smaller or equal to a certain number. From the figure, we can observe that, the violations are detected in 132 apps, and the majority of the apps had 3 or fewer violations. Specifically, the 74 true strong violations are from 31 apps, and the 267 true weak violations are from 101 apps.

Study on detection errors. Our default technique generated 3 false positives for strong violations and 58 false positives for weak violations. From our manual inspection, the cause for the majority of the false positives was that the privacy policies used phrases not in our ontology to describe the private information they collected. For example, a privacy policy used the phrase "carrier provider" to describe mobile network provider. Since the phrase is not in our ontology, our technique mistakenly determined that network information is not mentioned in the privacy policy and reported a false violation. Another relatively minor cause of false positives was that, in some cases, our NLP technique may have mistakenly filtered out paragraphs related to data collection. In such a scenario, the phrases in the removed paragraph would not be detected, so their corresponding API methods would not be added to $A_{represented}$ for the app and thus a violation would be raised.

[8]Processing of the rest 102 failed due to time-out, heap overflow, or other exceptions.

Table 1: Evaluation of Detected Violations

Approach	Type	# De	# True	# Mis	# FP
Default Technique	Strong	58	55	0	3
	Weak	344	267	19	58
	Total	402	322	19	61
SUSI Only	Strong	15	12	0	3
	Weak	82	44	31	6
	Total	97	56	31	9
Keyword Search	Strong	1	0	0	1
	Weak	389	261	74	54
	Total	390	262	74	55

Overall, our default technique was able to detect privacy-policy violations in a significant number of top Android apps and our false positive rate is relatively low, suggesting that developers do not need to waste much effort on inspecting false violations.

Comparison of variant techniques. To answer **RQ2**, we implemented the two variant techniques described in Section 5.1.3 and applied them to our data set. The three techniques detected 406 violations in total. Specifically, the Keyword-Search variant detected a proper subset of the violations detected by our default technique, and the SUSI-Only technique detected 3 weak violations and 1 strong violation that our default technique did not find. Our manual inspection found that 2 of the weak violations and the 1 strong violation were true positives. In Table 1, columns 3-6 presents the number of detected violations (# De), the number of correctly-classified true positives (# True), the number of misclassified true positives (# Mis), and the number of false positives (# FP), respectively.

From the table, we make the following observations. First, among the 3 techniques, our default technique was able to detect the most violations and achieved the highest type-classification accuracy (322/402 = 80.0%). Therefore, our default technique was more effective in general. Second, the SUSI-Only variant was only able to detect 97 violations, with 31 strong violations misclassified as weak violations. Inspection of the missed violations showed that the major reason for missed violations was that the SUSI categories are simply too coarsely grained. This was apparent particularly for the category of network information where an API method may be mapped to a phrase that has not much relation with it. For example, the method getNetworkOperatorName should be mapped to "carrier network", but under the umbrella of network information, it is also mapped to "Wifi Access Points", "MAC Address", etc. Therefore, an app that sends carrier network information through this API method may be interpreted as not having a related violation because of the mistakenly mapped phrases are in the privacy policy. Third, the keyword-search technique is able to detect slightly fewer violations, but it cannot classify strong violations from weak violations. The reason is that, the more abstract a phrase is, the more likely that it appears in a paragraph that is not related to data collection. For example, the phrase "MAC Address" is almost always used to describe data collected, while the phrase "network" may be used for many different purposes (e.g., "social network"). Since the keyword-search variant cannot filter out paragraphs that are not data-collection related, it can mistakenly extract many abstract phrases from irrelevant sections. Under the umbrella of these abstract phrases, an API methods can easily find a map, and thus a

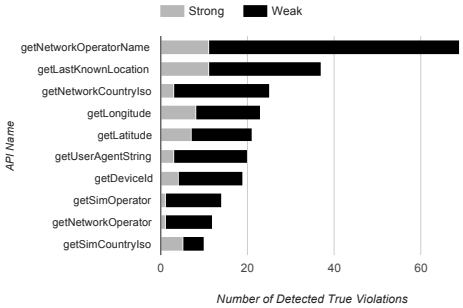


Figure 6: APIs with Most Detected Violations

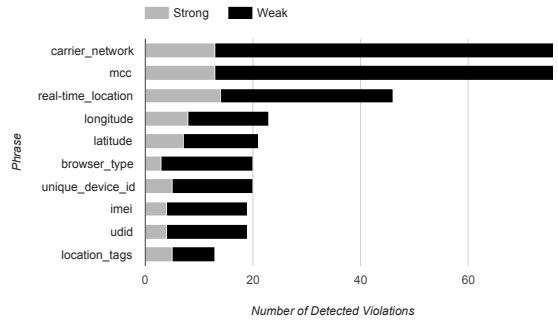


Figure 7: Terms with Most Detected Violations

strong violation is mistakenly identified as weak violation.

Top privacy information types in detected violations. To understand what types of private information are silently collected in privacy-policy violations, we studied the 10 API methods and phrases that were associated with most detected true violations. The results are presented in Figure 6 and Figure 7, respectively. In the two figures, the y-axis shows the name of the API or phrase, and the x-axis shows the number of strong (shown in gray) violations and weak (shown in black) violations associated with the API method or phrase. For brevity, we present only the short name of APIs in Figure 6.

From Figure 6, we can observe that the top API methods associated with detected violations fall into 4 major categories. The API methods ranked 1st, 3rd, 8th, 9th, 10th in the list are all regarding mobile network information such as carrier network and mobile country code. The API methods ranked 2nd, 4th, and 5th are about GPS location information. The other two relatively smaller categories are getUserAgentString (browser information) ranked the 6th, and getDeviceID ranked the 7th. Similarly, Figure 7 shows a similar trend in that mobile country code (mcc) and carrier network are the most common missing phrases. The category of GPS location is related to the phrases ranked the 3rd, 4th, and 5th. The following two categories are different ways to describe device identifiers such as IMEI, UDID, etc., and the browser type. It should be noted that location information is one of the most important types of information involved in detected violations since it is required to be explicitly stated in privacy policies [2].

5.3 Threats to Validity

Construction Validity is the extent to which we have measured what we think we are measuring [40]. In this study, we collected annotated phrases privacy policy terms and API terms that concern platform information. To address this threat, we provided annotators with the same process and instructions for identifying relevant phrases, and we selected only those policy terms where two or more workers agreed to the annotation. When constructing the ontology, the investigators employed heuristics [37] to justify the classification and agreement was measured using a chance-corrected statistic at each iteration to identify disagreements for reconciliation. With respect to measuring violations, we introduced two kinds of violations: strong violations wherein the known policy terms were not found in the associated policies, and weak violations in which the app's policy includes vague terms that are generally associated with the information accessed by the app.

Internal Validity refers to whether the causal inferences we derive from the dataset are valid [40]. When mapping policy terms to API terms, the investigators made numerous inferences. To reduce this threat, the investigators decomposed the mapping process into multiple, independent tasks: annotating the policy excerpts to identify personal information accessed through the platform, identifying ontological relationships between policy terms, identifying canonical names for platform API, and classifying those names by the policy terms in the ontology. For each of these step, the work was limited to small tasks in which each datum was individually reviewed by multiple investigators. The entire process consumed more than 33 hours, however, it reduced the likelihood that inferences would be missed, for example, by identifying relevant API method descriptions and aligning them directly with policies in an otherwise ad hoc fashion.

External Validity refers to the extent to which our results generalize to other policies and domains. In this study, we manually examined 50 policies to construct the ontology and policy-API mapping. To reduce this threat, we selected the most frequently used apps from different app categories to enhance the representativeness of our data set. However, we only sampled from Android apps, thus it is possible that our results do not extend to iOS-based apps, or further to web-based applications. However, some of our apps only had a combined privacy policy for mobile and web-based applications, thus, we are confident that our method could be extended to web-based applications with further research.

6. DISCUSSION

6.1 Violation Detection

We have shown in Section 5.2 that the mapping helps to facilitate policy violation detection with a reasonable number of false positives. This trait could be used to assist developers in verifying policy consistency for their own code. In such a scenario, the tool could bring to light potential violations between policy and code that may not be immediately or intuitively obvious to the end user or developer. Furthermore, because the framework would be used as a quality assurance tool for developers, false positives do not present a threat since, by definition, a falsely detected violation would be covered by the privacy policy.

Violation detection is improved through the use of our ontology (as relevant in the detection of strong and weak violations). The ability for compliance to be implied through transitive relationships between terms allows for the improvement of the overall approach without the need for more method-phrase maps. An increase in coverage could be achieved by simply improving or adding phrases to the ontology, resulting in the possible detection of more weak violations. The relationships between methods and policy phrases also have the potential to be applied to new and existing API documentation to improve privacy risk awareness. This could be used as a standard through which methods could be associated directly and indirectly to policy-oriented phrases. Such annotations could

then be used for violation detection and policy generation as well as an indication of the privacy-relevant information types produced by the method.

The potential benefits to developers from our work imply a need for a tool. We are currently working on an Android Studio IDE plugin, *PoliDroid*, that takes an existing privacy policy and source code as input and notifies the developer of potential violations in a similar style to syntax error highlighting. The tool works by scanning the privacy policy for phrases from our ontology and producing a list of permissible API method calls, $API_{represented}$. The tool then scans through the source code for API method calls that exist in our entire mapping to produce a set of methods that both exist in the code and that we have data for, API_{used}. If API_{used} contains methods not present in $API_{represented}$, a notification is raised signifying a potential violation. Effectively, the tool takes source code and a natural language policy to raise warnings about the terms used in the policy. The tool would also be able to function without a policy as input and instead produce terms with which a policy can be created. This would work by scanning the code for methods in our mapping and producing a minimum set of terms that would cover all of the methods. We believe such a tool would be appropriate for Android app developers due to their familiarity with the IDE.

6.2 Ontology Learning System

The ontology produced for this work was based only on phrases found in 50 privacy policies owing to the time-intensive nature of manually annotating the policies. To enable the inclusion of related phrases from many more policies into our ontology, we have developed an Ontology Learning System (OLS) based on Natural Language Processing (NLP) tools and the OWL API. Using this learning system we have been able to extract phrases with subsumption relationships between them from privacy polices. We ran this tool on a sample of 493 privacy polices and we were able to extract 783 sentences that are related to information collection. Moreover, the subsumption relationship between the phrases in the same sentence were also identified. Currently, we are working to improve the results of the tool through sanitization and enhancements to the parser. As a next step, we will use the OWL API to automatically extend the initial ontology used in this study with extracted phrases and relationships from the OLS.

6.3 Web-Based Applications

During our analysis of the privacy policies, we found that many of the policies were made to cover a wide range of apps not limited to AndroidOS. In most of these cases, the text implied the relevance of web-based applications. We believe that our technique has the potential to be applied to such applications. However, since web-based applications do not freely provide their implementation (such as Android's APK bytecode) it would be difficult to mine their API method calls. Furthermore, the wide variety of web application frameworks, languages, platforms, and servers would add more layers of implementation to take into account. This is out of our scope until there is enough data from web-based applications with common components.

7. RELATED WORK

Prior work exists on the factoring of privacy and privacy policies into source code. To our knowledge, ours is the only technique that works to bridge the gap between privacy policy and implementation through the use of natural language mappings to API methods.

7.1 Privacy and Permissions

Android has a permission system that is used for apps go to gain access to certain API methods. An app must declare these permissions as part of its source code. In turn, the user is notified during installation as to what the app requests. This permission system is related to privacy policies in that the methods that are accessed through the permissions are, or should be, represented in the app's privacy policy. An app's privacy policy can be cross-checked with an app's permissions, but permissions are not necessarily defined at method-level granularity. There is existing research that explores both privacy policies and OS permissions.

Rowen et al. have developed an IDE plugin, Privacy Policy Auto-Generation in Eclipse (PAGE), for generating privacy policies along side the development of the app [32]. PAGE works by guiding the user through a series of questions about the implementation of the app. Based on the answers, PAGE uses existing policy templates to generate a privacy policy for the app. Unlike our technique, PAGE does not take into account method calls or information flow and cannot be used for the detection of policy violations. As described in Section 6.1, we are working on a tool that uses our mappings to both generate phrases for use in privacy policies and verify the accuracy of a policy with respect to its app.

The static analysis tool PScout was developed by Au et al. for the analysis of the Android OS permission system [7]. The Android permission system helps policy consistency since it is used to show, at a course-grained level, the data that the app can access. PScout maps these permissions to method calls in order to evaluate their coverage. PScout itself does not work as a tool for linking policies to code, but its analysis of the Android permission system shows a limited interconnection of method-permission mappings with over 80% of methods related to only one permission. Among our mappings, 59% of the methods mapped to only one phrase. We believe this is due to how our approach uses canonicalization to expand a method's description to potentially associate it with more than one policy phrase. This higher rate of interconnectivity is important for violation detection since methods can return objects from which more than one type of information can be collected.

Existing work by Petronella presents a tool that relates natural language in privacy policies to Android permissions [28]. The tool works by providing the user with the list of permissions for an app along with the sentences from the privacy policy that are related to each permission. This is similar to our work in that it maps natural language phrases to potential data collection actions in the app's implementation. However, it is limited to the granularity of the Android permission system. Our work looks past permissions and related policy phrases directly to method invocations.

Stoaway, a tool created by Felt et al., detects over-privilege in compiled Android apps [16]. The tool found that among 940 apps, 35.8% were over-privileged. Analysis of the apps showed that copied code and testing artifacts were among the reasons for unnecessary privileges. These kinds of defects can be brought to light with policy verification through the use of our violation detection framework.

Vidas et al. have created a tool, Permission Check Tool, to assist developers in selecting minimal necessary permissions [36]. The tool uses static analysis to check the code for API references. Those references are cross-checked with a permission-method database created by the authors in order to generate a minimal set of permissions. These tools help to minimize privacy issues purely from an implementation standpoint. Our mappings can be used in conjunction with these to verify the corresponding policies.

Similarly, Bello-Ogunu et al. have created an Eclipse IDE plugin, PERMITME to guide developers in selecting Android permis-

sions [8]. The authors tested the tool on students in a mobile application development course and found that the assistance reduced the time spent on assessing privacy permissions and was found to be helpful and welcomed overall. PERMITME does not integrate natural language privacy policies in any way. However, the results of their study support the idea a privacy tool targeted at developers may help to alleviate the time spent on privacy compliance. Our data would allow for more privacy assurance with policy checking.

A new model is being used for the permission system in Android 6.0 [4] While the new model is backwards compatible with apps that use the old model, new apps will use new features that may affect previous work regarding the old permission model. Our framework is more robust in the sense that it is not affected by changes to the permission model since it works on the API level.

7.2 Security and Privacy Ontologies

Much research has been carried out on ontology implementation and usage in computer and information systems in recent years [10, 13, 18, 22, 35]. However, the majority of these works are regarding permission based systems, firewalls, and pervasive systems. We discuss them there.

Breaux et al. implemented an ontology to analyze the privacy policy of multi-tier systems to find the conflicts between the policies regarding data collection, usage, retention and transfer [12]. The authors consider Facebook, Zynga (the developers of Farmville), and the AOL advertising company which provides advertisements in Farmville in a case study identifying the conflicts between their privacy policies. This is done by mapping the privacy requirements from natural language policies to description logic in order to minimize ambiguity. An ontology of web application privacy policy terminology, similar to the ontology used in our research, is used for this work.

In other research, Chen et al. implemented a shared ontology for ubiquitous and pervasive applications called SOUPA [13]. Their research included modular component vocabularies to represent intelligent agents with associated beliefs, desires, and intentions, time, space, events, user profiles, actions, and policies for security and privacy. This research take advantage of spatial relations in reasoning about the collected information from user. They used the shared ontology to facilitate decision making in dynamic environment. This ontology does not consider the privacy policy related to the mobile applications and is mainly concerned about security policies in pervasive systems and knowledge sharing using the ontology.

In the domain of ontologies as access control and permission systems, Kagal and Finin considered client-server model and the privacy policies related to both web services and clients as users of web services [22]. Their ontology was implemented for use as an access control model and was implemented based on user specification and requirements for web services. Similarly, Grandon and Sadeh preserve users' contextual information from being misused by web-applications using an ontology as an access control mechanism [18]. In their research, users control who has access to their contextual information and under which conditions. These works focus on using ontologies as a mechanism for security, whereas the ontology used in our work is a tool for validation of intent.

Bradshow et al. implemented an ontology based on policies for behavior regulation of intelligent agents to continually adjust their behavior and maximize the effectiveness of intelligent systems [10]. The ontology is used for specification, management, conflict resolution, and enforcement of policies in intelligent systems which does not align with our goal in mobile applications privacy policies.

7.3 Information Flow Analysis

In our work, we leverage information flow analysis to check whether privacy information accessed from a certain method flows to the network. To achieve this, we used FlowDroid [6], a state-of-art static information analysis tool for Android apps. Other Android-oriented static information analysis techniques include CHEX [24], LeakMiner [39], and ScanDroid [17]. Specifically, CHEX detects component hijacking vulnerabilities in Android applications by tracking taints between externally accessible interfaces and sensitive sources or sinks. LeakMiner is an earlier context-insensitive information-flow analysis technique for detecting privacy leaks in Android apps. ScanDroid tracks information flows among multiple apps and detects privacy leaks into other apps. There are also dynamic information flow analysis techniques such as TaintDroid [15] and CopperDroid [34] that may detect privacy leak at runtime. It should be noted that all of the information-flow analysis techniques take a formal specification (e.g., a list of allowed flows from certain source methods to sink methods) and detect privacy leaking. However, the privacy policies are written in natural language so information-flow analysis cannot be directly applied. Our framework helps to bridge the gap by using the ontology and API method mappings, and may benefit from more advanced information flow analysis techniques in the future.

8. FUTURE WORK

Our work in this paper has revealed problems that expand beyond our original scope. First of all, privacy data transmitted through the network does not necessarily imply that the data is eventually stored (i.e., collected) in the organization's servers. To determine if the data that is written to the network is actually collected and stored, we plan to develop novel techniques to further trace the flow of information from the user's phone to the respective organization's data repositories. Second, when detecting weak violations, our framework does not distinguish between closely related phrases (e.g., "device information" and "device id") and distantly related phrases (e.g., "technology" and "browser language"). In the future, we plan to calculate semantic distance between phrases in the ontology in order to better measure the severity of weak violations. Third, a lot of privacy information of mobile app users is provided by users though the user interface of the apps. Therefore, besides Android API methods that are already considered in our framework, we plan to future extend our framework to handle user inputs as a new type of information source.

9. CONCLUSION

Mobile applications collect significant amount of privacy sensitive data from their users' devices. Organizations that develop these applications have a legal and moral obligation to clearly articulate to their users what data are being collected in the form of an application-specific privacy policy. However, these organizations lack sound mechanisms that can help them determine if the stated privacy policy is accurate—i.e., are the applications collecting pieces of privacy-sensitive data that is not stated in the policy? This paper presents a framework that addresses this problem by detecting such privacy policy violations in Android application code. Furthermore, its use has discovered 341 strong and weak violations from 477 top Android applications.

Acknowledgement

The authors are supported in part by NSF Awards CNS-0964710, CNS-1330596, CCF-1464425, NSA Grant on Science of Security, and DHS grant DHS-14-ST-062-001.

10. REFERENCES

[1] FTC report on Credit Karma and Fandango. https://www.ftc.gov/news-events/press-releases/2014/03/fandango-credit-karma-settle-ftc-charges-they-deceived-consumers, 2014.

[2] FTC report on Snapchat. https://www.ftc.gov/news-events/press-releases/2014/06/ftc-testifies-geolocation-privacy, 2014.

[3] Developer economics q1 2015: State of the developer nation. https://www.developereconomics.com/reports/developer-economics-q1-2015/, 2015.

[4] Permissions. https://developer.android.com/preview/features/runtime-permissions.html, 2015.

[5] Smartphone os market share, q1 2015. http://www.idc.com/prodserv/smartphone-os-market-share.jsp, 2015.

[6] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 259–269, 2014.

[7] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228. ACM, 2012.

[8] E. Bello-Ogunu and M. Shehab. Permitme: integrating android permissioning support in the ide. In *Proceedings of the 2014 Workshop on Eclipse Technology eXchange*, pages 15–20. ACM, 2014.

[9] J. Bhatia and T. Breaux. Towards an information type lexicon for privacy policies. In *8th IEEE International Workshop on Requirements Engineering and Law (RELAW)*, pages 19–24, 2015.

[10] J. Bradshaw, A. Uszok, R. Jeffers, N. Suri, P. Hayes, M. Burstein, A. Acquisti, B. Benyo, M. Breedy, M. Carvalho, et al. Representation and reasoning for daml-based policy and domain services in kaos and nomads. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 835–842. ACM, 2003.

[11] T. Breaux and F. Schaub. Scaling requirements extraction to the crowd: Experiments on privacy policies. In *22nd IEEE International Requirements Engineering Conference (RE'14)*, pages 163–172, 2014.

[12] T. D. Breaux, H. Hibshi, and A. Rao. Eddy, a formal language for specifying and analyzing data flow specifications for conflicting privacy requirements. *Requirements Engineering*, 19(3):281–307, 2014.

[13] H. Chen, F. Perich, T. Finin, and A. Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pages 258–267. IEEE, 2004.

[14] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20:37–46, 1960.

[15] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6, 2010.

[16] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, 2011.

[17] A. P. Fuchs, A. Chaudhuri, and J. S. Foster. Scandroid: Automated security certification of android applications. *Manuscript, Univ. of Maryland, http://www.cs.umd.edu/avik/projects/scandroidascaa*, 2(3), 2009.

[18] F. L. Gandon and N. M. Sadeh. Semantic web technologies to reconcile privacy and context awareness. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):241–260, 2004.

[19] J. Godfrey and C. Bernard. State of the app economy 2014. 2014.

[20] M. Grüninger and M. S. Fox. Methodology for the design and evaluation of ontologies. 1995.

[21] K. D. Harris. *Privacy on the Go: Recommendations for the Mobile Ecosystem*. 2013.

[22] L. Kagal, T. Finin, M. Paolucci, N. Srinivasan, K. Sycara, and G. Denker. Authorization and privacy for semantic web services. *Intelligent Systems, IEEE*, 19(4):50–56, 2004.

[23] P. G. Kelley, L. F. Cranor, and N. Sadeh. Privacy as part of the app decision-making process. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3393–3402. ACM, 2013.

[24] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang. Chex: Statically vetting android apps for component hijacking vulnerabilities. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 229–240, 2012.

[25] C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[26] S. Matsumoto and K. Sakurai. A proposal for the privacy leakage verification tool for android application developers. In *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*, page 54. ACM, 2013.

[27] S. Papadopoulos and A. Popescu. Privacy awareness and user empowerment in online social networking settings. http://www.computer.org/web/computingnow/archive/january2015, 2015.

[28] G. Petronella. *Analyzing Privacy of Android Apps*. PhD thesis, Politecnico di Milano, 2014.

[29] S. Rasthofer, S. Arzt, and E. Bodden. A machine-learning approach for classifying and categorizing android sources and sinks. In *2014 Network and Distributed System Security Symposium (NDSS)*, 2014.

[30] J. R. Reidenberg, J. Bhatia, T. D. Breaux, and T. B. Norton. Automated comparisons of ambiguity in privacy policies and the impact of regulation. *Journal of Legal Studies*, 2016.

[31] J. R. Reidenberg, T. Breaux, L. F. Cranor, B. French, A. Grannis, J. T. Graves, F. Liu, A. M. McDonald, T. B. Norton, R. Ramanath, et al. Disagreeable privacy policies: Mismatches between meaning and usersŠ understanding. 2014.

[32] M. Rowan and J. Dehlinger. Encouraging privacy by design concepts with privacy policy auto-generation in eclipse (page). In *Proceedings of the 2014 Workshop on Eclipse*

Technology eXchange, pages 9–14. ACM, 2014.

[33] J. Saldana. *The Coding Manual for Qualitative Researchers*. SAGE Publications, 2012.

[34] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro. Copperdroid: Automatic reconstruction of android malware behaviors. In *22nd Annual Network and Distributed System Security Symposium*, 2015.

[35] A. Uszok, J. M. Bradshaw, J. Lott, M. Breedy, L. Bunch, P. Feltovich, M. Johnson, and H. Jung. New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of kaos. In *Policies for Distributed Systems and Networks, 2008. POLICY 2008. IEEE Workshop on*, pages 145–152. IEEE, 2008.

[36] T. Vidas, N. Christin, and L. Cranor. Curbing android permission creep. In *Proceedings of the Web*, volume 2, 2011.

[37] S. Wadkar and T. Breaux. Towards an information ontology for personal privacy. Technical report.

[38] T. Warren. Google touts 1 billion active android users per month. `http://www.theverge.com/2014/6/25/5841924/google-android-users-1-billion-stats/`, 2014.

[39] Z. Yang and M. Yang. Leakminer: Detect information leakage on android with static taint analysis. In *Proceedings of the 2012 Third World Congress on Software Engineering*, pages 101–104, 2012.

[40] R. Yin. *Case Study Research: Design and Methods*. SAGE Publications, 2013.

[41] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang. Vetting undesirable behaviors in android apps with permission use analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 611–622. ACM, 2013.

[42] L. X. Zhao. Privacy sensitive resource access monitoring for android systems. Master's thesis, Rochester Institute of Technology, 2014.