Inferring Semantic Mapping Between Policies and Code: The Clue is in the Language

Pauline Anthonysamy[1,2], Matthew Edwards, Chris Weichel, and Awais Rashid

[1] Google Switzerland
anthonysp@google.com
[2] Security Lancaster, Lancaster University, UK
{p.anthonysamy, m.edwards7, c.weichel, a.rashid}@lancaster.ac.uk

Abstract. A common misstep in the development of security and privacy solutions is the failure to keep the demands resulting from high-level policies in line with the actual implementation that is supposed to operationalize those policies. This is especially problematic in the domain of social networks, where software typically predates policies and then evolves alongside its user base and any changes in policies that arise from their interactions with (and the demands that they place on) the system. Our contribution targets this specific problem, drawing together the assurances actually presented to users in the form of policies and the large codebases with which developers work. We demonstrate that a mapping between policies and code can be inferred from the semantics of the natural language. These semantics manifest not only in the policy statements but also coding conventions. Our technique, implemented in a tool (CASTOR), can infer semantic mappings with F1 accuracy of 70% and 78% for two social networks, Diaspora and Friendica respectively – as compared with a ground truth mapping established through manual examination of the policies and code.

1 Introduction

This paper addresses the problem of identifying areas of code that operationalize (or implement) one or more policy statement(s) from security or privacy policies. This problem is particularly challenging because information systems have grown not only in size and technical complexity but also in the volume of information they manage and process. The effort required to identify areas of code that implement relevant policies remains largely manual, at best aided by simple search techniques. Ideally, policies and code should be linked to ease processes such as compliance checks, verification, maintenance etc.; however this is not always the case for two main reasons:

(i) Asynchronous evolution of policies and code. Policies describe organisations' actions on user data or personally identifiable information – and are often driven by regulatory and legal requirements. Program code, on the other hand, implements the various features and services provided by the information system and must be compliant with the aforementioned policies. Modern information systems evolve rapidly as organisations continually update the system's

functionality to provide a better quality of service and user experience. This is generally driven by factors such as changes in requirements, optimisation of code, fixes for bugs and security vulnerabilities, etc. Policies also change but such changes are less frequent and often driven by legislative requirements and regulatory frameworks or changes in business processes. This asynchronous evolution can often (unintentionally) lead to changes resulting in the code being non-compliant with the policy. A recent example is that of Facebook introducing a photo sync feature that allows users to sync their mobile photos with their Facebook account [5]. This feature introduced a vulnerability that allowed photos that had not been published on Facebook and should not have been visible to anyone be accessed by third-party applications; yet Facebook's terms of service continued to stipulate that private photos will stay private when connecting to external applications.

(ii) Implementation precedes policies and regulation. In an ideal world, policies would be derived first, requirements established, and then passed on to software engineers for design and implementation. However, much modern software development does not follow this cycle. They also, almost always, out-pace the regulatory environment. Often, legal and regulatory requirements are not given full consideration during product development (requiring post-implementation compliance checks) or regulations come into existence *after* a system is in public use. For example, the European Commission only recently introduced regulation as part of its Data Protection Directive [4] requiring that users should have full export/download access to all of the data stored about them.

In this paper we present a technique and tool to infer and identify areas of code (aka functions) that implement particular policy statements described in natural language. Our inference technique is driven by the semantics of natural language and coding conventions, wherein verbs and nouns used in policy statements and source code (e.g., in function and parameter names) provide useful clues that enable a *semantic mapping* to be established between the two artefacts. Our use of naming conventions means that such mapping can be added to systems post-hoc – as we highlighted above, it is often impossible to attach security demands arising from high-level policies to methods at the time the code is written (e.g., because of codebases predating policies).

Contributions

We make the following novel contributions in this paper:

1. We describe a *semantic-mapping* approach to infer function specifications from natural language policies. The resulting technique aids developers in inferring and identifying relevant functions that implement one or more policy statement(s) to assist in compliance verification. Our technique demonstrates that the burden of identifying areas of code that operationalize relevant policies can be reduced through inference using the semantic constructs of the natural language itself and coding conventions driven by such constructs.
2. An implementation of our technique in a tool called Castor is presented. It accepts as inputs policy statements and source code; and outputs a set

of semantic mappings between policy statements and function specifications (methods). These mappings aid one to assess the completeness of an implementation with respect to stated policies. More importantly, the semantic mapping which is established *directly* between policy statements (as presented to users) and source code deem useful for organisations in quality assessment and compliance preservation.

3. We present an evaluation of our technique and tool on inferring mapping between privacy policies and the code implementing these policies for two open-source social networking sites, namely Diaspora and Friendica. Our evaluation shows that we can achieve a F1 accuracy of 70% for Diaspora and 78% for Friendica (for the balanced class experiment) in finding the semantic mappings required as compared with a ground truth mapping established by thorough manual examination of the policies and code.

2 Related Work

In this section we first contrast our work with techniques that automate mappings between textual documents and source code, followed by an approach to privacy leak detection using flow analysis. We then discuss techniques that automate mappings between policies and software requirements specifications.

Text Documents and Source Code: Pandita et al. [24] attempt to transform natural language descriptions of methods, as found in API documentation, into formal specifications for function behaviour, as described by *code contracts*. Their method involves parsing the API documentation through Part-Of-Speech (POS) tagging aided by domain-specific noun boosting and jargon handling, followed by the application of a shallow parser which attempts to classify the sentences of lexical tokens based on predefined semantic templates. The result of this process is a first-order logic expression which is then parsed for equivalences and redundancies and finally used to generate code contracts [3]. These contracts can then be inserted into the functions to which the corresponding API documents refer. While their work demonstrates the possibility of mapping between natural language and representations of source code, it differs from our work in two ways: firstly, we are interested in identifying implementations of policy statements in the source code, rather than in generating assertions for error–checking; and secondly, the mapping is done in a far more narrow domain than attempted in this paper, as API documentation is naturally more precisely connected to the source code it describes than user-facing texts such as policies.

Antoniol et al. [9, 10] describe an approach to establish and maintain links between source code and free text documents such as requirements, design documents and user manuals, etc. Their work is based on the assumption that programmers use meaningful names for program concepts, such as functions, variables, types, classes, and methods; therefore, the analysis of these concepts (identifiers) can aid in associating high level concepts with program concepts, and vice-versa. The approach is based on a stochastic language model that assigns a probability value to every string of words taken from a prescribed vocabulary.

The relevant documents are used to estimate the language models, one for each document or identifiable section. Then, a classifier – Bayesian classification – is used to compute the score of the sequence of mnemonics extracted from a selected area of code against the language models. A high score indicates a high probability that a particular sequence of mnemonics is extracted from the document, or a section, that generated the language model. This implies the existence of a semantic link between the document and the area of code from which the particular sequence of mnemonics is extracted. However, the approach is primarily applied to text that is likely to clearly express source code functionality (requirements specification documents). In contrast, our approach addresses the scenario where code and policies have either evolved independently or policies have come into existence post-development and deployment.

Privacy in Source Code: Jang et al. [18] approach the breach of privacy by user-facing websites through flow analysis of the Javascript code from several major websites. Their method involves the design and implementation of a language for specification of privacy-breaching information flows, and was trialled on a large sample of well-visited websites. Where their approach tests uniformly for four specific information breaching flows to identify violations, we aim to tie source code to the publicly expressed policies of social networking sites. Our approach also performs analysis of source code, but whereas their method analyses client-side code, we perform analysis of the server-side handling of information, which is arguably more critical for tracing potentially hidden violations.

Policies and Requirements: Massey et al. [21] evaluated the security and privacy requirements of an existing software system – the iTrust, open source electronic health record system – for legal compliance with a regulatory document (HIPAA). Their work mainly focuses on establishing trace links between software requirements and legal texts which, while an important initial step in legal compliance, does not fully complete the mapping between legal texts such as policies and the software code itself. Cleland-Huang et al. [14] proposed two machine learning methods to automatically generate links between regulatory codes (a subset of HIPAA) and product requirements. May et al. [22] present a framework that formalises regulatory rules, HIPAA, and exploit this formalisation to analyse the rules' conformance to a health-care system automatically. Fisler, et al. [16] also attempt a model-checking based verification system, Margrave, for analysing role-based access control policies. However, these works focus on deriving software requirements from privacy policies and legal documents (primarily in the healthcare domains). In contrast, we aim to establish a semantic mapping between areas of code (functions) that implement particular policy statements described in natural language – and in situations where policies need to be mapped to code post-hoc implementation.

3 Semantic Inference

Castor's semantic inference mechanism (cf. Appendix A for Castor's architecture) presents a technique that enables software developers to infer and identify

areas of code (aka functions) that implement particular policy statements described in natural language. This inference technique is driven by the semantics of natural language and coding conventions, wherein *verbs* and *nouns* used in policy statements and source code (e.g., in *function* and *parameter* names) provide useful clues that enable a *semantic mapping* to be established between the two artefacts.

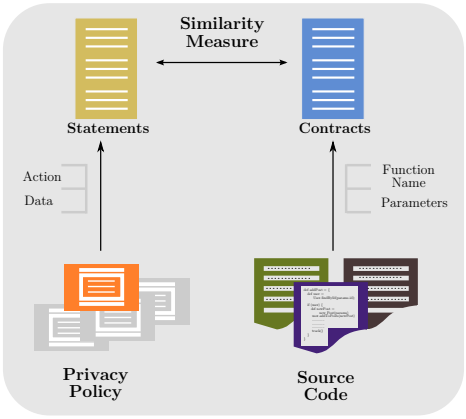

Fig. 1. An overview of our semantic mapping approach.

A premise of this work is that programmers use meaningful names for source code primitives, such as functions, parameters and classes. Much of the application domain knowledge that developers employ when writing code is often captured by these mnemonics for code primitives; thus these mnemonics aid in associating source code primitives with high-level concepts (e.g., policy statements) [10]. In this section, we provide basic definitions and concepts relating to policies, source code and the relationship between policy and source code primitives, wherein, we measure how close the code primitives, namely **functions** and **parameters**, are to the policy primitives of **actions** and **data**.

3.1 Definitions

The semantic mapping between policy and source code is drawn using a semantic relatedness measure between the primitives of these artefacts, namely, similarity between the words e.g., *data-parameter*, *action-function*, etc. As summarised below, we define a model for privacy policies, source code functions and followed by the *semantic relatedness* between the two. Herein,

– A policy, $\mathcal{PP}$, is considered to be a set of statements. Each statement $s \in \mathcal{PP}$ is modelled as the tuple $s = \langle a, D \rangle$, where $D = \{d_1, d_2, ..., d_n\} \subseteq \mathcal{D}$ is the data items referred to by the statement and $a \in A$ is the action verb (e.g., share, track, collect, etc.).

- $\mathcal{F}$ is the set of functions implemented in source code. A function $f = \langle c, n, P \rangle, f \in \mathcal{F}$ is modelled as a triple, where c is the class to which it belongs, n is the function's name and P is the set of its parameter names.
- Semantic relatedness, $\mathcal{R} : \mathcal{W} \times \mathcal{W} \rightarrow [0, 1]$ is the measure of semantic similarity between two words $w_0, w_1 \in \mathcal{W}$.
- $\mathcal{M}_F : \mathcal{S} \rightarrow \mathcal{W} \subseteq \mathcal{F}$ is a relationship between policy statements, $s \in \mathcal{S}$, and source code functions, $f \in \mathcal{W}$, where $\mathcal{W}$ is the subset of functions that map to one or more statements in $\mathcal{S}$. This relationship is computed using the semantic relatedness measure defined above, applied to the words used in policy statements and function/parameter names.

In the following subsections each of the above modelling and mapping techniques are elaborated.

3.2 Policy Model Construction

The construction of the policy model, $\mathcal{PP}$, is based on two common linguistic analysis techniques, namely part-of-speech tagging and shallow parsing. Part-of-Speech (POS) tagging is the process of assigning parts of speech, such as noun, verb, adjective, etc., to each word in a text (statements). A shallow parser accepts the lexical tokens generated by the POS tagger and divides those tokens into segments which correspond to certain syntactic units, such as noun phrases, verbs, verb phrases, etc. Figure 2 illustrates a simplified example of a parsed policy statement.

The annotated statements are mapped based on their grammatical functions to policy primitives of 'action', the activity that the actor performs and 'data', the data item to which an actor's action relates. In doing this, the fact that each grammatical function has a designated semantic role in natural language is exploited. Actions, for example, are expressed by any of the verbs or verb phrases (VP) in natural language, while data tends to be identified by nouns and noun phrases. For example, the tokens labelled [**VB:** post] and [**NN:** post] in Figure 2 will be tagged as *action* and *data* respectively.

[**WRB:** Whenever] [**PRP:** you] [**VB:** post] [**JJ:** content] [**IN:** like] [**NN:** status] [**NNS:** updates] [**PRP:** you] [**MD:** can] [**VB:** select] [**DT:** a] [**NN:** privacy] [**VBG:** setting] [**IN:** for] [**DT:** every] [**NN:** post].

Legend: **WRB:** Whadverb, **PRP:** Personal pronoun, **VB:** Verb, **JJ:** Adjective, **IN:** Preposition, **NN:** Noun (singular), **NNS:** Noun(plural), **MD:** Modal, **DT:** Determiner, **VBG:** Verb

Fig. 2. An example of tagged policy statement.

This grammatical mapping process is aided by a data dictionary to assist when mapping composite data primitives such as *'personally identifiable information'*. The data dictionary is used to associate and identify relevant noun phrases with pre-defined data classes. Without this, 'personally identifiable information' would be annotated as an adjective phrase by the POS tagger instead

of as a noun phrase as required for this analysis. This association is essential to the semantic mapping step in which such composite data primitives are expanded to obtain the individual data items that are grouped within that class such as 'gender', 'sexuality', 'relationship status', etc. Note, the POS Tagger used here was adapted from the Stanford Parser [19].

To aid in retaining the core elements of the policy statements, i.e., verbs and nouns, selected terms and grammatical constructs are removed. These include stop words (e.g., *the*, *is*, *at*, *when*, etc.), personal and possessive pronouns. The decision to retain only the core elements of the policy statement is to construct an intermediate policy model that is easy to comprehend and allows for cohesion with the original statement. Although, formalised policies like P3P [15] and EPAL [11] have been proposed to make policies more readable and enforceable, they have several limitations, e.g., the P3P language does not have a clear semantics and can therefore be interpreted and presented differently by different user agents; and, an EPAL policy must be enforced at the time data is accessed which causes significant performance overhead – every data access has to rely on an external policy evaluation. Furthermore, the policy model proposed in this work avoids the additional complexity that comes with formalisation and utilises the semantics of natural language constructs which can be interpreted and translated appropriately.

3.3 Source Code Model Construction

The source code model is constructed automatically using a (naive) static program analysis technique [26]. The analyser parses the code base of an online social network and constructs a model based on class, functions and parameter names. This model is inspired by code contracts [23] which are a way of abstractly expressing what a function accomplishes. Functions, $\mathcal{F}$, are modelled as triples, $f = \langle c, n, P \rangle$, where c is the *class* to which the function belongs, n is the *function's name* and P the set of its *parameter names*. Note: in this paper the terms 'parameter' and 'variable' are used interchangeably. These code principles are extracted to ease the semantic mapping (described next) of policy and source code primitives.

3.4 Semantic Mapping

The semantic mapping, $\mathcal{M}_F$, between $\mathcal{S}$ and $\mathcal{F}$ is based on the premise that policy statements are operationalized as functions at the source code level. The strategy for establishing this semantic mapping is based on a hybrid approach of Natural Language Processing (NLP) and machine learning applied to policy statements and source code. We use a lexical resource, namely WordNet[3] to discover the semantic relatedness, $\mathcal{R}$, (a measure of "similarity") between policy and source primitives. WordNet is a broad coverage lexical network of English words that contains around 100,000 terms, organised into taxonomic hierarchies. Nouns, adjectives, verbs and adverbs are organised into networks of synonym

[3] `http://wordnet.princeton.edu/`

sets (synsets) that each represent one underlying concept and are interlinked with a variety of relations. For instance, a word that has multiple meanings (polysemous) will appear in one sysnset for each of its definitions. The measure of relatedness between two words, $w_0, w_1 \in \mathcal{W}$, in WordNet is computed using path length in the network graph: $\mathcal{R} : \mathcal{W} \times \mathcal{W} \rightarrow [0, 1]$. The shorter the path from one word to another, the more similar they are.

We then use a machine learning technique to map statements to functions using the computed similarity measures (input to the machine learning algorithm). The trained classifier can then distinguish between a correct and incorrect mapping when it is confronted with new similarity values by using the learned mapping model.

Examination of Naming Conventions: As previously mentioned, the semantic mapping approach is drawn from the concept of relating policy and source primitives. We measure how close the source primitives (variables/parameters or functions) are to the policy primitives of actions and data. Common programming practices tend to dictate that functions are named as verbs and variables are named as nouns [25]. These naming conventions are crucial to this approach, so we verified whether this practice held in the real world. A unigram POS tagger from the Python Natural Language Toolkit[4] was run across the source code from two social networks, Diaspora[5] and Friendica[6]. These two code bases are the datasets used for evaluation in this paper.

The tagger was trained on Brown corpus[7] (a general text collection that contains 500 samples of English-language text, totalling roughly to one million words), with a regular expression based backoff parser implementing a technical dictionary. We ran the tagger over a collection of function and variable names drawn from the source code of Diaspora and Friendica. As the common `camelCase` and `snake_case` coding conventions are likely to confuse a natural language tagger, such examples were split into their individual words (e.g, `camelCase` to `camel case`).

As shown in Table 1, while function parameters mapped as expected to nouns (77.50%), results vary for the function name mapping. Unsplit function names were mostly categorised as nouns by default, but splitting these names into constituent tokens revealed a modest increase in the proportion of tokens identified as verbs. Further examination showed that the first token after such splitting was in most cases a verb, as in `get_name` or similar constructs. 44.2% of function names contained at least one verb token. The relatively high parameter-to-noun and function-to-verb semantic relatedness illustrates that the approach for data-to-parameter and action-to-function mappings is a viable measures in terms of drawing a similarity between policy and source primitives.

[4] `http://nltk.googlecode.com/svn/trunk/doc/howto/wordnet.html`
[5] `https://github.com/diaspora`
[6] `https://github.com/friendica/friendica`
[7] `http://www.essex.ac.uk/linguistics/external/clmt/w3c/corpus_ling/content/corpora/list/private/brown/brown.html`

Table 1. Verb and Noun percentages of function names and variables.

	% Nouns	% Verbs	# Tagged
parameters	77.50	6.82	21034
parameters (split)	75.35	8.23	27967
function name	68.04	25.68	5366
function name (split)	56.48	27.89	11842
function name (first token)	43.20	44.20	5366

Mapping Inference: The problem of mapping policy statements to source code functions that operationalize those statements is formulated as a binary classification problem, because the mappings are either correct or incorrect. Our semantic inference is an application of the Random Forests [12] classifier, which is an effective approach to the problem of learning and classification [20, 17]. We found that this classifier best fitted our mapping model and outperformed other standard classifiers such as naîve bayes [1] and support vector machine [2]. The classifier needs to be trained once per social network (domain-dependent), as random forests are a supervised learning technique. This is performed using manually created mappings. By confirming the manually mapped $s-f$ pairs, one can then provide more training data to the classifier and improve its prediction.

To infer the mapping, for each policy statement $s \in \mathcal{S}$ the classifier predicts if a source code function $f \in \mathcal{F}$ maps to that statement, that is $\langle s, f \rangle \in \mathcal{M}_F$. And, to do this, labelled examples, i.e., a training dataset of correct and incorrect mappings, are required to estimate a 'target learning model' in the machine learning technique. This estimated learning model is then used to classify an input vector of features into classes. In CASTOR, the labelled examples are generated using manually created mappings. These manually created mappings are established based on a method that was derived in prior work [8]. The method provides a systematic means of studying the traceability (mapping) between privacy policies and controls in social networks, hence establishing the *degree of traceability* between the two. In [8], we define the degree of traceability as the level of certainty that we can have about the existence of an *externally observable relationship* measured using a qualitative 3-point scale.

By confirming the manually mapped statement/function pairs, one can then provide more training data to the classifier and improve its prediction (target learning model). We label such manual mapped $\langle s, f \rangle$ pairs as G (indicating correct mappings), while non-mapping pairs are labelled N (indicating incorrect mappings). For each statement, function pair $\langle s, f \rangle$ we extract a feature vector $\mathbf{v} = \langle dc, af, dp, pc \rangle$ for classification:

1. **data-class-similarity**, $dc = \mathcal{R}_{\max_{d \in \mathcal{D}_s}}(d, c_f)$, where $\mathcal{D}_s$ is the set of data items of the statement s, and c_f is the class name of the function f;
2. **action-function-similarity**, $af = \mathcal{R}_{\max_{a \in A_s}}(a, n_f)$, where A_s is the set of actions of the statement s, and n_f is the name of the function f;

3. **data-parameter-similarity**, $dp = \mathcal{R}_{\max_{d \in \mathcal{D}_s, p \in P_f}}(d, p)$, where $\mathcal{D}_s$ is the set of data items of the statement s, and P_f are the parameter names of the function f;
4. **parameter count**, $pc = |P_f|$ is the number of parameters of function f.

The WordNet path similarity is used as a measure for semantic relatedness of the feature variables dc, af, and dp. When actions, data items, parameter names or function names consist of multiple words W, the maximum similarity of these words were used as semantic relatedness: $\mathcal{R}(W, x) = \max_{w \in W}(w, x)$.

The measure of semantic relatedness as outlined above generates a set of vector of features for the learning method, which classifies each vector of features into the set of mapping classes, $V = \{G, N\}$. For example (for the training dataset), the feature vector for the statement–function pair $\langle s_1, f_6 \rangle$ shown below (see Statement s_1 & Listing. 1.1) is $\mathbf{v} = \langle 0.67, 0.00, 0.74, 5 \rangle$. Thus, in order to calculate the most probable class (G or N) for this vector, the features are run down all of the trees in the forest and the final class of the vector is decided by aggregating the votes (i.e., predicted class) of each tree – which is G in this case.

Statement, s_1: The default privacy setting for some of the information you post on Diaspora is set to "everyone".

```
def setDefault
    #note :id references a postvisibility

    @post = Post.where(:id => params[:post_id]).select("id, guid, author_id").first
    @contact = current_user.contact_for(@post.author)

    if @contact && @vis = PostVisibility.where(:contact_id => @contact.id,
                                               :post_id => params[:post_id]).first
      @vis.hidden =! @vis.hidden
      if @vis.save
        render 'update'
        return
      end
    end
    render :nothing => true, :status => 403
  end
end
```

Listing 1.1. Snippet of function, f_6, `setDefault` from the Diaspora code base.

4 Evaluation

The data used in our experiments consists of privacy policies and source code of two social networks: Diaspora and Friendica. Both of these sites are decentralised social networks implemented using Ruby on Rails and PHP respectively. We selected these sites in accordance with the following constraints: availability of source code (open-source), at least 1000 function specifications, and the fact that they are implemented using different programming languages and frameworks. The motivation behind this selection is to test the coverage of our semantic mapping technique across different conventions used in real–world implementations.

Since the two social networks are decentralised open source networks, there were no publicly available privacy policies. This is a constraint that we faced since most popular social networks with a published privacy policy are closed source systems. We, therefore, synthesised policies drawing upon our earlier detailed investigation of privacy policies of 16 social networks [7], in which we showed that there exist a significant disconnect between policy statements and user-facing privacy controls. The synthesised privacy policies were representative of those that would be shown to users of these sites[8].

This section describes the different (independent) experiments conducted using machine learning techniques for the semantic inference. Recall that for each experiment the input to the classifier is the set of pairs $\langle s, f \rangle$ where each pair consists of the features $\mathbf{v} = \langle dc, af, dp, pc \rangle$. The results of these experiments and the conclusions drawn are then presented.

4.1 Experiment 1: Unbalanced classes

There was a drastic imbalance of classes in our experimental datasets. Non-mapping statement–function pairs (class N) are far more common than mapping ones (class G) – see Table 2. This is due to the inherent nature of our input, there are significantly more contracts that are not relevant to policy statements compared to those that are relevant. In this unbalanced experiment we train the classifier on this unbalanced data, but adjust the weights of the class importance during learning, so that the equal error rate $EER = |FPR - FNR|$ is minimised (FPR is the false positive rate, FNR is the false negative rate).

Table 2. Class imbalance $\|N\| \cdot \|G\|^{-1}$ for all 2 datasets, with and without heuristics.

Dataset	No Heuristics	Heuristics	% Reduction
Diaspora	1347.01	601.98	44.69
Friendica	2195.99	700.81	31.91

For each dataset we manually created a ground truth mapping (based on the method in [8]). We trained network-dependent classifiers using an 80/20 training/test data split which we evaluated using a randomised cross validation. We report scores based on true positive rates (recall), TPR, false positive rates, FPR, precision, PPV and $F1$ score. The recall score for each class, namely G and N, provides information on the number of semantic mappings that were successfully identified, while the precision score takes into account all identified mappings for each class and evaluates how many of them were actually relevant. Finally, the F1 score is the harmonic mean of precision and recall (see Appx. B).

4.2 Experiment 2: Balanced classes

A common practice for dealing with imbalanced data sets is to rebalance them artificially. This is essential to evaluate the fundamental soundness of our semantic

[8] See example policies at `http://www.paulineanthonysamy.com/myData.html`

mapping approach. Over and under-sampling methodologies have received significant attention as a technique to rebalance classes [13]. Therefore, in the second experiment we trained and tested CASTOR's classifier on balanced datasets. For each dataset (one for each of the two social networks) we balance both classes (G, N), by randomly sampling an equal number of statement/function pairs. This random resampling method for balancing classes has been shown to be an effective technique when faced with an imbalance problem [13] as in our case.

$$\underset{s \in \mathcal{PP}, f \in \mathcal{F}}{\text{rand}} \langle s, f \rangle \text{ s.t. } |\langle s, f \rangle \in \mathcal{M}| = |\langle s, f \rangle \notin \mathcal{M}|.$$

4.3 Experiment 3: Introducing heuristics

To alleviate the class imbalance, we introduce heuristics that exclude source code functions that are unlikely to map to policy statements. An expert would expect operationalizing functions to be located in specific places (i.e. packages and folders), depending on the programming language and framework that was used to implement the social network. We encode that knowledge and reject functions based on where in the source code they are defined. Below are some of the heuristics introduced:

- Global: Sources within the `'db/'`, `'spec/'`, `'config/'`, `'lib/'`, `'script/'`, `'markdown/'` folders across our dataset were removed. These folders were selected as they contain database table descriptions, application wide configuration files, third-party library files, scripts and markdown files.
- Framework Specific: These were mainly to deal with the different terminologies and spellings among the folders.
  - PHP: Sources within the `'view/'`, `'util/'`, `'test/'`, `'mods/'`, `'library/'` folders were removed.
  - Ruby: Sources within the `'presenters/'`, `'assets/'`, `'views/'`, `'mailers/'`, `'error_message'`, `'layout'` folders were removed.

Our heuristics do not reject functions that were manually labelled as ground truth. This way we reduce the class imbalance by 44.69% for Diaspora and 31.91% for Friendica respectively across the two social networks (see Table 2).

4.4 Results

Figure 3 depicts a box plot of mean similarity scores obtained from WordNet for ground truth (mapped), G, and non-mapping, N, statements respectively – indicated by the center horizontal line within each box. The outliers are represented by •. The scores are computed for each statement, function pair $\langle s, f \rangle$ with dc, af, dp, pc. As illustrated by Figure 3 the mean scores for the two sites are higher for the ground truth (mapped) statements, namely 0.343 (s.d. 0.220) for Diaspora and 0.346 (s.d. 0.208) for Friendica. In comparison the non-mapping statements' means were 0.146 (s.d. 0.117) for Friendica and 0.166 (s.d. 0.127) for Diaspora. These values show that, although the overall similarity scores are

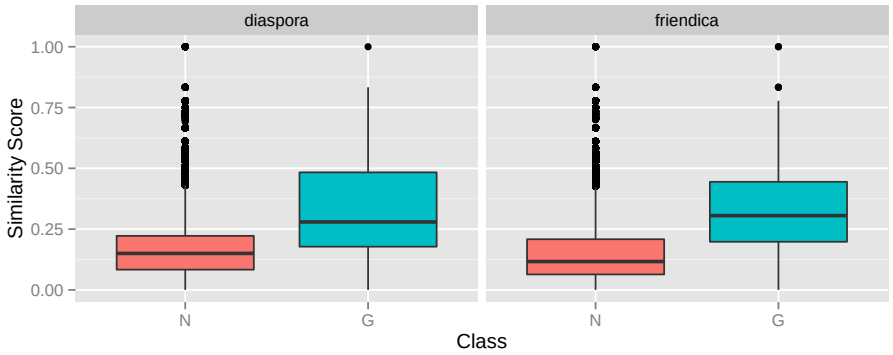

Fig. 3. Mean similarity scores of ground truth, G, and non-ground truth mappings, N.

Table 3. Table showing results from Random Forest classifier. The table labels are as follows:- Recall: TPR, False Positive Rate: FPR, Precision: PPV, F1 score: $F1$, Equal Error Rate: EER.

Dataset	TPR	FPR	PPV	F1	EER
Diaspora					
Balanced	0.693	0.296	0.700	0.696	0.011
Unbalanced	0.759	0.265	0.002	0.004	0.024
Heuristic	0.777	0.301	0.004	0.008	0.078
Friendica					
Balanced	0.788	0.245	0.762	0.775	0.033
Unbalanced	0.806	0.242	0.001	0.003	0.048
Heuristic	0.790	0.315	0.003	0.007	0.105

small, WordNet consistently returned a higher similarity score for statements in G than statements in N, which warrants that our semantic mapping approach achieves its aim as to infer the mapping between policy statements and code.

The semantic mapping results are reported in Table 3 for all three experiments: unbalanced, balanced, and with heuristics. In all instances, the recall (TPR) rates were consistently high for Diaspora (between 0.69 and 0.78) and Friendica (between 0.79 and 0.80) indicating a high level of success in the identification of semantic mappings for each of our classes – G and N. These rates are crucial as it illustrates that our approach works in the non-optimal case, i.e., unbalanced classes, which is the norm in the real world. The consistent TPR and FPR rates shows that our approach generalises, and performs well, over different social networks. The EER (representing the number of false positive and false negative are equal) were also consistently low across all the experiments – at an average of 5% and 6% for Diaspora and Friendica.

We observe a very low precision in the unbalanced experiment (0.002). This is to be expected as it has been observed previously [13] that class imbalance (i.e., significant differences in class sizes) may produce a deterioration of the performance achieved by learning and classification systems. This precision score (PPV) significantly improved when the class sizes were balanced (Diaspora: 70% and Friendica: 76%).

Introducing simple heuristics to the unbalanced class improved precision (by a mean factor of 2.19, s.d. 0.24). Albeit a small increase, the observed improvement was proportional to reduction of the class imbalance shown in Table 2. This indicates that using heuristics improves the classification performance.

5 Discussion & Future Work

The scale and complexity of current systems make the task of identifying relevant sections of code (functions) that implement or realise a policy extremely challenging. Our technique demonstrates that this burden of identifying areas of code that operationalizes relevant policies can be reduced through inference (**F1 accuracy** of **70%** and **78%** for Diaspora and Friendica – balanced class experiment) using the semantic constructs of the natural language itself and coding conventions driven by such constructs. Though the functionality of a method is most critical in ensuring that requirements are upheld, this mandates that security demands arising from high-level policies are explicitly attached to methods at the time the code is written. This is infeasible nigh impossible in typical scenarios where code bases predate policies. Our approach allows this connection to be made based on well-established naming conventions. While this would never be as precise as a detailed semantic analysis of each method's code, the latter would be extremely expensive. Our usage of naming conventions means that such mapping can be easily added (post-hoc) to systems to highlight methods, which may need to be checked against security demands arising from policies. By identifying and short-listing the relevant methods, our approach not only benefits developers but potentially policy or compliance auditors for data sensitive systems such as Facebook and Google that are prone to accidental breaches.

Limitations: Our semantic mapping approach relies on WordNet's similarity measures to compare policy and source code primitives. The overall WordNet similarity scores are low as it is designed as a dictionary based on psycholinguistic principles rather than a knowledge base. WordNet lacks contextual policy information. For example, in a social-networking policy, WordNet does not interpret 'track' as 'recording information' therefore we were compelled to take the most-related pair of synsets among the matched options. We hypothesized that these measures can be significantly increased if a verb-synonym database was available and later confirmed it [6]. The verb synonym database was built by extracting all the verbs from the privacy policies analysed in [8] and manually classifying them based on their semantic meanings. The semantic meanings of these verbs were determined using a lexical dictionary.

Table 4. Table showing results from Random Forest classifier with the verb synonym database. The table labels are as follows:- Recall: TPR, False Positive Rate: FPR, Precision: PPV, F1 score: $F1$, Equal Error Rate: EER.

Dataset	TPR	FPR	PPV	F1	EER
With Verb Synonym Database					
Diaspora					
Balanced	0.785	0.251	0.757	0.771	0.031
Unbalanced	0.735	0.250	0.002	0.004	0.013
Heuristic	0.762	0.245	0.006	0.011	0.017
Friendica					
Balanced	0.797	0.209	0.792	0.795	0.006
Unbalanced	0.836	0.230	0.002	0.003	0.066
Heuristic	0.806	0.281	0.004	0.008	0.087

The results of the three experiments improved when conducted with a verb synonym database (cf. Table 4). In particular, the recall rates (TPR) increased for both datasets – Balanced: Diaspora: 78.5% and Friendica: 79.7%; Unbalanced: Diaspora 73.5% and Friendica 83.6%; and, Heuristic: Diaspora 76.2% and Friendica 80.6%. The precision also increased for both the datasets in the balanced class experiment, i.e., 77.1% and 79.5% accordingly. Although, the precision score was still relatively low in the unbalanced and heuristic experiments (due to the fact that the classes were still vastly disproportionate), there was a small hike in Friendica's PPV rates – 0.1% rise – but there was no change in Diaspora. Whereas, the heuristic experiment improved the PPV rates for both datasets, i.e., about 0.2% in Diaspora and 0.1% in Friendica.

Acknowledgements

This research was funded by Lancaster University 40th Anniversary Research Studentship and has no ties to the first author's current employment at Google.

A Implementation: CASTOR

We have implemented our technique in a tool called Castor. Figure 4 illustrates the architecture of Castor. Castor accepts as inputs policy statements and source code; and outputs a set of semantic mappings between policy statements and functions. Briefly, Castor works on the input as follows:

Policy Engine: Castor's policy engine is composed of a parser and a statement analyser which transforms the natural language policy into an intermediate representation (as described in Section 3.2). This intermediate representation

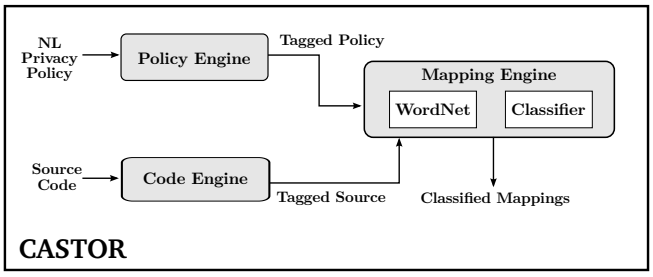

Fig. 4. CASTOR's Architecture.

maintains the relevant policy primitives of a statement, namely action (verbs) and data (nouns).

Code Engine: CASTOR's code engine is composed of a minimal recursive-descent parser that extracts a function's name, associated class and parameters, along with information identifying the source file and line number where the function can be found. This is inline with our source model construction in Section 3.3.

Mapping Engine: CASTOR's mapping engine infers the mapping between the privacy policy $\mathcal{PP}$ and source code functions $\mathcal{F}$ using its inbuilt WordNet corpora and classifier. The output of this engine is a set of semantic mappings between policy statement(s) and functions.

B Formulae

Recall (TPR) $= \frac{tp}{tp+fn}$; *False-Positive Rate* (FPR) $= \frac{fp}{fp+tn}$; *Precision* (PPV) $= \frac{tp}{tp+fp}$; and $F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$.

References

[1] Naive bayes. `http://www.nltk.org/_modules/nltk/classify/naivebayes.html`
[2] SVM. `http://www.nltk.org/_modules/nltk/classify/svm.html`
[3] Code contracts (2010), `http://research.microsoft.com/en-us/projects/contracts/`
[4] EU data directive 95/46/ec (February 2014), `http://eur-lex.europa.eu/`
[5] Facebook photo leak flaw raises security concerns (March 2015), `http://www.computerweekly.com/news/2240242708/Facebook-photo-leak-flaw-raises-security-concerns`
[6] Anthonysamy, P.: A Framework to Detect Information Asymmetries between Privacy Policies and Controls of OSNs. Ph.D. thesis, Lancaster University (2014)
[7] Anthonysamy, P., Greenwood, P., Rashid, A.: Social networking privacy: Understanding the disconnect from policy to controls. IEEE Computer (June 2013)
[8] Anthonysamy, P., Greenwood, P., Rashid, A.: A method for analysing traceability between privacy policies and privacy controls of online social networks. In: Preneel, B., Ikonomou, D. (eds.) Privacy Technologies and Policy, Lecture Notes in Computer Science, vol. 8319. Springer Berlin Heidelberg (2014)

[9] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Tracing object-oriented code into functional requirements. In: Program Comprehension, 2000. Proceedings. IWPC 2000. 8th International Workshop on. pp. 79–86 (2000)
[10] Antoniol, G., Canfora, G., de Lucia, A., Casazza, G.: Information retrieval models for recovering traceability links between code and documentation. In: Proceedings of the International Conference on Software Maintenance (ICSM'00). IEEE Computer Society, Washington, DC, USA (2000)
[11] Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise Privacy Authorization Language (EPAL). Tech. rep., Rschlikon (2003)
[12] Breiman, L.: Random forests. Machine Learning 45 (2001), http://dx.doi.org/10.1023/A%3A1010933404324
[13] Chawla, N.V., Japkowicz, N., Kotcz, A.: Editorial: Special issue on learning from imbalanced data sets. SIGKDD Explor. Newsl. 6(1), 1–6 (Jun 2004)
[14] Cleland-Huang, J., Czauderna, A., Gibiec, M., Emenecker, J.: A ML approach for tracing regulatory codes to product specific requirements. In: ICSE (2010)
[15] Cranor, L., Langheinrich, M., Marchiori, M.: A p3p preference exchange language 1.0 (appel 1.0). World Wide Web Consortium, Working Draft WD-P3P-preferences-20020415 (April 2002)
[16] Fisler, K., Krishnamurthi, S., Meyerovich, L.A., Tschantz, M.C.: Verification and change-impact analysis of access-control policies. In: Proceedings of the 27th International Conference on Software Engineering. pp. 196–205. ICSE '05, ACM, New York, NY, USA (2005)
[17] Haiduc, S., Bavota, G., Oliveto, R., De Lucia, A., Marcus, A.: Automatic query performance assessment during the retrieval of software artifacts. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. pp. 90–99. ASE 2012, ACM, New York, NY, USA (2012)
[18] Jang, D., Jhala, R., Lerner, S., Shacham, H.: An empirical study of privacy-violating information flows in javascript web applications. In: Proceedings of the 17th ACM Conference on Computer and Communications Security. pp. 270–283. CCS '10, ACM, New York, NY, USA (2010)
[19] Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1. pp. 423–430. ACL '03, Stroudsburg, PA, USA (2003)
[20] Ma, L., Torney, R., Watters, P., Brown, S.: Automatically generating classifier for phishing email prediction. In: Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on. pp. 779–783 (Dec 2009)
[21] Massey, A., Otto, P., Hayward, L., Antn, A.: Evaluating existing security and privacy requirements for legal compliance. Requirements Engineering (2010)
[22] May, M.J., Gunter, C.A., Lee, I.: Privacy apis: Access control techniques to analyze and verify legal privacy policies. In: Proceedings of the 19th IEEE Workshop on Computer Security Foundations. pp. 85–97. CSFW '06, IEEE Computer Society, Washington, DC, USA (2006)
[23] Meyer, B.: Object-Oriented Software Construction. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edn. (1988)
[24] Pandita, R., Xiao, X., Zhong, H., Xie, T., Oney, S., Paradkar, A.: Inferring method specifications from natural language api descriptions. In: Proceedings of the 34th International Conference on Software Engineering. ICSE '12 (2012)
[25] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.E., et al.: Object-oriented modeling and design, vol. 199. Prentice Hall (1991)
[26] Wagner, D.: Static analysis and computer security: New techniques for software assurance. Ph.D. thesis, University of California at Berkeley (December 2000)