PriGuardTool: A Web-Based Tool to Detect Privacy Violations Semantically

Conference Paper · May 2016

DOI: 10.1007/978-3-319-50983-9_5

2 authors, including:

Pinar Yolum
Bogazici University
136 PUBLICATIONS **1,465** CITATIONS

SEE PROFILE

PriGuardTool: A Web-based Tool to Detect Privacy Violations Semantically

Nadin Kökciyan (✉) and Pınar Yolum

Department of Computer Engineering,
Bogazici University, 34342 Bebek, Istanbul, Turkey
{nadin.kokciyan,pinar.yolum}@boun.edu.tr

Abstract. Online social networks contain plethora of information about its users. While users enjoy sharing information online, not all information is meant to be seen by the entire network. Managing the privacy of users has become an important aspect of such online networks. An important part of this is detecting privacy violations and notifying the users so that they can take appropriate actions. While various approaches for detecting privacy violations exist, most of the approaches do not have a running tool that can exhibit the principles of its underlying approach. This paper presents PriGuardTool, a Web-based tool that can detect privacy violations in online social networks. Each user is represented by a software agent in the system that first collects user's privacy concerns, explicitly specified as what types of content are meant to be seen by which audience. The system represents these privacy constraints as commitments between the user and the online social network. The user constraints are converted into commitments automatically by the agent. The system then monitors which commitments are violated based on the content shown to users, such that a violated commitment represents a privacy violation in the system. While checking for violations, the effects of posts on the system as well as the semantic relations and rules are considered. We evaluate PriGuardTool by using various real-life scenarios and real data that have been collected over Facebook. Our initial results show that realistic privacy violations can be detected using PriGuardTool.

Keywords: Privacy, Online Social Networks, Commitment, Ontology

1 Introduction

Privacy is the right of an individual to express herself selectively. An individual may prefer to expose certain information about herself to a certain group of others, but may choose to hide another set of information. This right is difficult to maintain on the Web since information can propagate easily. It is even worse on online social networks since different users can share content about an individual, without expecting an explicit confirmation from the individual. This results in tremendous privacy violations to take place [5].

Consider the following examples: A user herself misconfigures the system and reveal unintended content (e.g., the user shares holiday pictures with colleagues when not intending to); or a friend of a user shares a content not knowing that the user would not want the content online (e.g., a friend shares a picture where the user is drunk). These simple examples show that both a user herself or friends can take simple actions that lead to privacy violations. More importantly, sometimes the privacy violations are more subtle. In order to be discovered, they require various pieces of information to be put together. For example, a user does not reveal her location but shares a picture that has an embedded geotag. Any software that can process the geotags can help others discover the user's location [6]. Sometimes, the information needed to decipher the violations is not that straightforward. For example, looking at two friends check-ins to a remote island could signal that they are together. Inferring this information, when neither have explicitly specified it, could easily violate their privacy. In all cases, the users seek tools that will help them to preserve their privacy and catch privacy breaches if any, so that they can take an action.

Most of the existing commercial systems on the Web allow a user to specify constraints on her own posts only and enforce them. However, this does not necessarily avoid privacy violations. That is, if a user does not want her colleagues to see her holiday pictures but her group holiday picture is shared publicly by a friend, her privacy is still violated. Various approaches to deal with privacy violations exist in the literature. One set of approaches aim to prevent privacy violations in the first place [15]. The approaches that employ argumentation or negotiation techniques among users to reach agreements before sharing content fall into this category [25,17,9]. Another set of approaches aim to detect privacy violations. This set of approaches represent user's privacy constraints formally and try to find out if the network evolves into a state where these constraints are violated. An important work is that of Hu *et al.*, where privacy concerns are represented as multiparty access control rules [7]. Their work is based on a social network model, a multiparty policy specification scheme and a mechanism to enforce policies to resolve multiparty privacy conflicts. They benefit from Answer Set Programming (ASP) to represent their proposed model. Another important work is that of Carminati *et al.* that studies a semantic web based framework to manage access control in OSNs by generating semantic policies [3]. Their proposed social network operates according to agreed system-level policies. In a similar line, we have previously proposed a semantic meta-model for representing agent-based online social networks [12]. We further proposed a model PriGuard that represents privacy constraints as commitments between users and the online social network, which are widely-used constructs for modeling interactions between agents [26]. This paper formally describes PriGuardTool, a Web-based tool that implements PriGuard model that detects privacy violations and notifies users to take an action.

PriGuardTool is a privacy management system. It enables users to enter their privacy constraints and then to check for privacy violations at desired times, similar in principle to virus checks. The privacy constraints capture a

user's expectation from the network; e.g., a user may not want her colleagues to see her pictures but might be fine if friends see them. A privacy violation can happen explicitly (e.g., if the user shares a picture with colleagues by mistake) or implicitly (e.g., if the actions of the user or the others lead colleagues to have access to the picture). PriGuardTool is equipped to check for both kinds of privacy violations. Implicit violations are especially difficult to detect because they require inferences to be made. To deal with this, PriGuardTool uses ontologies to represent knowledge and semantic rules, then it can check for violations both on the ontologies and on the inferred knowledge. To demonstrate its workings, we have implemented it so that it can work on real data that are extracted from Facebook. That is, a user can login with her Facebook credentials, allow the tool to download all her data (which are converted to an ontology), and check for privacy violations there.

The rest of this paper is organized as follows: Section 2 explains our approach for detecting privacy violations in online social networks. Section 3 develops the principles behind our developed tool. Section 4 explains in detail the design choices made to implement the proposed tool. Section 5 evaluates the tool in the context of a real online social network. Section 6 discusses the work in relation to other approaches for managing users' privacy in online social networks, and several limitations of current work.

2 Background: PriGuard Approach

PriGuard is a commitment-based model for privacy-aware online social networks [12] that enables users to detect privacy violations. Each user in the social network is represented by an agent that is responsible for keeping track of the user's privacy constraints and checking for violations when needed. The online social network is defined by the set of relationships, content types and a set of semantic rules. The set of relationships pertains to the users (e.g., friend, colleague, and so on). The content types capture allowed contents (e.g., picture, check-in, text, and so on). The semantic rules capture how the social network operates (e.g., content can be reshared, contents are shown selectively, and so on). A snapshot of the online social network captures the agents, their relationships and the content in the online social network. Even for a single snapshot, one can focus on different views of it. For example, the global view of the system would contain all the content of all the agents in the system; while a smaller, local view can contain the content shared by a single agent last month. PriGuard uses views to check if privacy of users are violated or not. Depending on the view, the extent of privacy check can be managed.

In PriGuard, the social network domain is formally defined using Description Logics (DL). The domain consists of concepts (e.g., `Agent`), relations (e.g., *isFriendOf*) and individuals names (e.g., `:alice`)[1]. On the other hand, there is a

[1] We denote a `Concept` with text in mono-spaced format, a *relation* with italic text, and an `:individual` with a colon followed by text in mono-spaced format.

need for semantic rules so that a social network can operate accordingly. In PriGuard, the semantic rules are specified as Datalog rules. These Datalog rules capture the fundamental operations of OSN, independent from specific users. For example, *sharesPost*(X,P) $\rightarrow$ *canSeePost*(X,P) is a Datalog rule, which states that an agent can see the posts that it shares. In this rule, *sharesPost* and *canSeePost* are predicate symbols; X and P are universally quantified variables.

While the OSN has its operation rules, the users have privacy expectations from the system. These privacy expectations are captured with conditional commitments [22,26]. Informally, conditional commitments represent a contract between two parties, such that each party commits to realizing certain predicates. In terms of privacy, this maps to a situation where a user commits to specifying information about friends, colleagues, and so on correctly and the OSN commits to ensuring the correct set of individuals will be shown the content (based on user's specification). A commitment is denoted as a four-place relation: *C(debtor*; *creditor*; *antecedent*; *consequent)*. The *debtor* is committed to the *creditor* to bring about the *consequent* if the *creditor* brings about the *antecedent*. First, each user specifies her privacy concern. Second, the agent transforms such a privacy concern into a commitment so that it can later be verified. Consider the following example:

Example 1 Charlie shares a concert picture with everyone and tags Alice in it. However, Alice does not want other users to see her pictures.

Here, Alice has a privacy concern such that she does not want to be seen by others. She does not have any control on what is shared by her friends. After Alice specifies her concern, Alice's agent (`:alice`) generates a commitment between `:alice` and the social network operator. The antecedent of the commitment describes the individuals affected by the commitment (e.g., agents except Alice) and the content that the commitment is about (e.g., pictures). The consequent of the commitment says whether the specified individuals should see or not see the content. Overall, Alice only promises to share content on this online social network if the online social network promises not to reveal information about her whereabouts.

A commitment violation occurs, when the antecedent holds but not the consequent. In another words, the social network operator fails to bring about the consequent. This signals a breach of privacy. To detect such commitment violations, each agent computes under which conditions a commitment would be violated and generates a violation statement. For example, people seeing Alice's pictures would violate Alice's commitment that is in an active state (i.e., the antecedent is achieved).

Agents use the domain information, the semantic rules, the view information and the violation statements to detect privacy violations. Then, each agent reports the detection results; it depends on the creditor of the commitment (e.g., the user) to take an action accordingly.

3 PriGuardTool

PriGuardTool is a Web-based tool that implements PriGuard model [12]. We use ontologies to capture the domain, view, and the semantic rules of the social network.

Domain: The social network domain is represented using PriGuard ontology specified in OWL 2 Web Ontology Language [18]. PriGuard model is a DL model, which can be completely defined in an OWL 2 ontology. In this ontology, there are classes that define domain concepts, object properties that relate individuals and data properties that describe individual-specific properties. For example, `Agent` is a class, which describes a set of users in the social network. `:alice` and `:bob` might be individuals that are elements of `Agent` class. These two individuals can be connected to each other via the object property *isFriendOf*. `:alice` can have a name "Alice Kingsleigh", which is described by the data property *hasName*.

In a social network, it is important to model the users, the relationships between users and the posts being shared by the users. As mentioned before, users are represented as `Agent` individuals in the ontology. The relationships are defined as object properties between agents. *isConnectedTo* is the most general property that defines a connection between two agents. However, it is possible to describe more specific relationships such as *isColleagueOf*. `Post` class is the most general class to represent a post. In an ontology, one can define complex classes. For example, we use complex classes to model specific posts such as `LocationPost`, which can be defined as: `Post` $\sqcap \exists$*hasLocation*.`Location` (posts that have at least one location). Each post is initialized by an agent (*hasCreator*). Moreover, an agent can share posts (*sharesPost*) and see posts (*canSeePost*). A post can be about an agent (*isAbout*). Posts can include textual, visual or locational information represented as `Text`, `Medium` and `Location` respectively. Mediums can include geotags (*hasGeotag*). A `Post` is related to these classes via *hasText*, *hasMedium* and *hasLocation* properties. A person can be mentioned in a text (*mentionedPerson*), tagged in a medium (*taggedPerson*) or at a location (*withPerson*). Each post can be associated with contextual information (`Context`) as well. A specific `Audience` is meant to see a post. *hasAudience* relates audience individuals to post individuals. Hence, members of this audience are described by the use of *hasMember* property.

View: In PriGuard ontology, a view is a set of class assertions (e.g., ClassAssertion(`Agent :alice`)) and object property assertions (e.g., ObjectPropertyAssertion (*isFriendOf* `:alice :charlie`)). In Table 1, we show the system view for Example 1. We use the abbreviations CA and OPA for ClassAssertion and ObjectPropertyAssertion respectively. The view of Example 1 is specified in functional-style syntax. At this particular view, `:charlie` creates and shares a post (`:pc1`) including a medium (`:picConcert`), an `:audience` with `:alice`, `:bob`, `:diane` as members and a person tag of `:alice`. The relationships are defined as follows:

Table 1. System View of Example 1. `:charlie` Shares a Post `:pc1`

CA(`Agent :alice`)	CA(`Agent :bob`)
CA(`Agent :charlie`)	CA(`Agent :diane`)
CA(`Post :pc1`)	CA(`Picture :picConcert`)
CA(`Audience :audience`)	
OPA(*isFriendOf* `:alice :bob`)	OPA(*isFriendOf* `:alice :charlie`)
OPA(*isFriendOf* `:bob :charlie`)	OPA(*isFriendOf* `:bob :diane`)
OPA(*sharesPost* `:charlie :pc1`)	OPA(*hasAudience* `:pc1 :audience`)
OPA(*hasMedium* `:pc1 :picConcert`)	OPA(*taggedPerson* `:picConcert :alice`)
OPA(*hasMember* `:audience :alice`)	OPA(*hasMember* `:audience :diane`)
OPA(*hasMember* `:audience :bob`)	OPA(*hasCreator* `:pc1 :charlie`)

`:alice`, `:bob` and `:charlie` are friends of each other; `:diane` is a friend of `:bob`. The remaining assertions include the class assertions for each instance.

DL Rules: A social network needs a set of semantic rules to operate. Recall that, in PriGuard, rules are defined as Datalog rules. OWL 2 is an expressive language to represent some Datalog rules as DL rules. For example, consider the rule r_7 in Table 2. This rule states that a post that includes a geotagged picture is an instance of `LocationPost` class in the ontology. The remaining DL rules are as follows. If an agent shares a post, then the agent can see it (r_1). An agent can see a post if it is in the audience of that post (r_2). If an agent creates a post then this post is about that agent (r_3). Similarly, a post is about an agent if the agent is tagged at a specific location (r_4), in a medium (r_5) or mentioned in a text (r_6). If an agent is tagged in a picture and shares another post by declaring its location then the location information of other agents tagged in that picture is revealed as well (r_8).

Table 2. Example Semantic Rules as Description Logic (DL) Rules

r_1:	$\mathit{sharesPost} \sqsubseteq \mathit{canSeePost}$
r_2:	$\mathit{hasMember}^- \circ \mathit{hasAudience}^- \circ \mathit{R\_sharedPost} \sqsubseteq \mathit{canSeePost}$
r_3:	$\mathit{hasCreator} \sqsubseteq \mathit{isAbout}$
r_4:	$\mathit{hasLocation} \circ \mathit{withPerson} \sqsubseteq \mathit{isAbout}$
r_5:	$\mathit{hasMedium} \circ \mathit{taggedPerson} \sqsubseteq \mathit{isAbout}$
r_6:	$\mathit{hasText} \circ \mathit{mentionedPerson} \sqsubseteq \mathit{isAbout}$
r_7:	$\texttt{Post} \sqcap \exists \mathit{hasMedium}.\exists \mathit{hasGeotag}.\texttt{Location} \sqsubseteq \texttt{LocationPost}$
r_8:	$\mathit{R\_locPost} \circ \mathit{sharesPost}^- \circ \mathit{taggedPerson}^- \circ \mathit{hasMedium}^- \circ \mathit{sharesPost}^- \sqsubseteq \mathit{isAbout}$

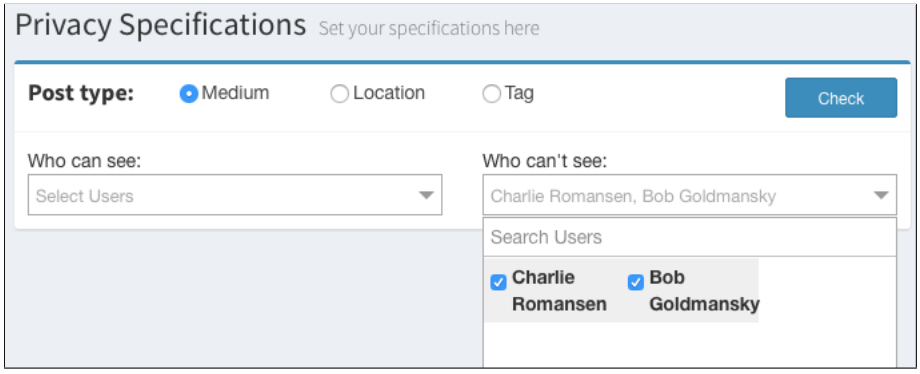

Fig. 1. Alice declaring her friends to not see her medium posts.

Commitments: Users input their privacy concerns via PriGuardTool interface as depicted in Figure 1. The user can specify her privacy concerns regarding medium posts, location posts and posts that the user is tagged in. For each category, the user declares two groups of people: one group that can see that category and a group that cannot. If the user specifies conflicting privacy concerns (e.g., a user is part of both groups), the agent adopts a conservative approach to minimize privacy violations to occur; i.e., it finds conflicting users and move them to the group that cannot see the content.

PriGuard ontology is used to semantically describe the commitments. Table 3 shows the commitments that Alice is in involved in. Since current OSNs are centralized, our commitments are among users and the OSN operator (`:osn`). Recall that Alice wants to be the only one who can see her medium posts (see Example 1). Hence, two commitments are generated C_1 and C_2. In C_1, `:osn` promises `:alice` to show her medium posts to `:alice`. In C_2, `:osn` promises `:alice` to not reveal her medium posts to others.

Table 3. Commitments for Example 1

C_i	<Debtor;	Creditor;	Antecedent;	Consequent>
C_1:	<`:osn`;	`:alice`;	X==`:alice`, *isAbout*(P, `:alice`), `MediumPost`(P);	*canSeePost*(X, P)>
C_2:	<`:osn`;	`:alice`;	`Agent`(X), not(X==`:alice`), *isAbout*(P, `:alice`), `MediumPost`(P);	not(*canSeePost*(X, P))>

Violation Statements: After all the semantic inferences are made by the use of PriGuard ontology and DL rules, the agent should be able to query this knowledge to detect privacy violations in the social network. A violation statement is a statement wherein a commitment would be violated. Here, agents use SPARQL queries to represent commitment violations. In another words, a violation statement is mapped to a SPARQL query.

SPARQL is a way of querying RDF-based information [21]. Note that ontological axioms can also be seen as RDF triples. In a SPARQL query, there are query variables, which start with a question mark (e.g., ?x), to retrieve the desired results. We only focus on *SELECT* queries with filter expressions *NOT EXISTS* and *EXISTS* to represent violation statements. Recall that the antecedent of a commitment includes information about agents that are the target audience of the commitment, and the set of posts being shared. The consequent of a commitment specifies whether agents could see or not the content. In the antecedent, each predicate of arity two is mapped into a RDF triple. For example, *isAbout*(P, `:alice`) is transformed into "?p osn:isAbout osn:alice". Each predicate of arity one is mapped into an *rdf:type* triple. For example, `Agent`(X) is transformed into "?x rdf:type osn:Agent". Equality or non-equality expressions become FILTER expressions in SPARQL. For example, not(X==`:alice`) is transformed into "FILTER (?x != osn:alice)". The consequent of a commitment is mapped into a FILTER EXISTS or FILTER NOT EXISTS expression in SPARQL. If the consequent of a commitment is positive, then this commitment is violated if the consequent does not hold and the antecedent holds; i.e., it is mapped to FILTER NOT EXISTS expression. Otherwise, it is transformed into a FILTER EXISTS expression. For example, the consequent of C_2 is not positive (not(*canSeePost*(X, P))) hence it is transformed into "FILTER EXISTS { ?x osn:canSeePost ?p }".

A complete SPARQL query is shown in Table 4. The *PREFIX* declares a namespace prefix. *osn* prefix shows where to find PriGuard ontology for querying. This *SELECT* query declares two query variables (?x and ?p) to be retrieved. The core part of the query is defined in the *WHERE* block, which consists of four triples (one is used in a filter expression). This query returns the set of posts that can be seen by agents except Alice.

`:charlie` shares a post `:pc1`, which includes a picture of `:alice` and `:charlie`. The audience is set to everyone. `:alice` checks for possible privacy violations. PriGuardTool finds the corresponding commitments: C_1 and C_2. C_1 is not violated since Alice can see her posts. However, the violation statement of C_2 (as shown in Table 4) holds in the system with the substitutions $\{?x/\{$`:bob`, `:charlie`$\}\}$ and $\{?p/$`:pc1`$\}$. Here, a privacy violation occurs because Alice and Charlie have conflicting privacy concerns; i.e, one wants to keep it personal while the other prefers sharing it with everyone. Thus, it is not possible to fulfill both of their concerns at the same time.

Table 4. The Violation Statement of C_2

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX osn: <http://mas.cmpe.boun.edu.tr/ontologies/osn#>

SELECT ?x ?p WHERE {
        ?x rdf:type osn:Agent .
        ?p osn:isAbout osn:alice .
        ?p rdf:type osn:MediumPost .
        FILTER EXISTS {
                ?x osn:canSeePost ?p .
                FILTER (?x != osn:alice)
        }
}
```

4 Implementation

We have implemented PriGuardTool as a Web application[2]. We have used PHP for the front-end development and Java for the back-end development. PriGuardTool is able to work with various social networks. For this, a gateway should be developed for user authentication and data collection. Here, we decided to work with Facebook since it is widely used around the world. We integrated *Facebook Login* to our web application to enable user authentication. We also implemented a Facebook gateway to collect data from Facebook users.

Figure 2 shows the information flow of PriGuardTool. The tasks are represented as rectangles. A human task is depicted as a task with a figure on top while the other tasks are automated tasks. The solid arrows represent the flow between tasks. The data operations are shown as dashed arrows. First, the user logs into the system by providing her Facebook credentials. The tool collects the user data and stores in a database (MongoDB). The user inputs her privacy concerns, which are stored as a JSON document. These privacy concerns are transformed into commitments between the user and the social network (Facebook) operator, and the corresponding violation statements (SPARQL queries) are generated as well. On the other branch, *Generate Ontologies* task takes care of reading user data from MongoDB, creating and storing ontologies in MongoDB. *Detect Privacy Violations* task uses SPARQL queries and the user's ontologies to monitor the social network for privacy violations. Finally, the user is shown a list of posts that violate her privacy if any. Then, the user can take an action such as modifying a post (e.g., removing a person from the audience of that post). Once the user logs out from the system, the tool removes the user data and the generated ontologies. This ensures that no information remains in the database after the detection is completed.

Data Collection: We extract information about the user from Facebook by the use of Facebook Graph API[3]. We request the following login permissions: *email*,

[2] `http://mas.cmpe.boun.edu.tr/priguardtool`

[3] `https://developers.facebook.com/docs/graph-api`

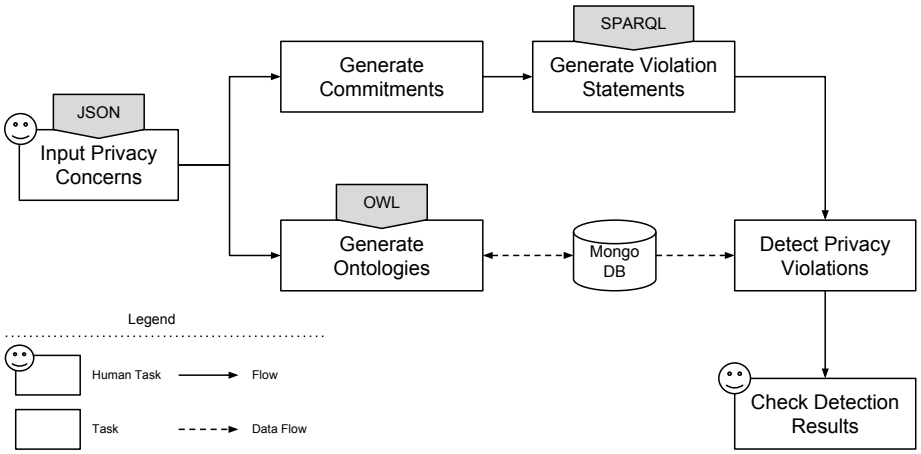

Fig. 2. PriGuardTool Implementation Steps

public_profile, *user_friends*, *user_photos*, *user_posts*. These permissions allow us to collect information about Facebook posts together with the comments and likes of other users. We use MongoDB[4], which is an open-source document-oriented database, to keep the extracted information. Graph API supports the exchange of JSON documents, and it becomes reasonable to store the user data as a JSON document in MongoDB. Note that we only extract information of the user, which may be shared by the user itself or by a friend of the user (i.e., the user is tagged in a post shared by a friend).

Facebook Graph API (v2.5) enables extraction of some information of a user, such as the user's posts, the comments on the posts or the likes of the posts. However, it does not allow us to extract some important information about the users, such as the list of friends of a user. Further, it is not possible to extract any information about the posts of other users. As another limitation, one cannot extract information about user-defined lists (e.g., if the user has a family list, it is not possible to get users that belong to that list). We analyze the collected information of the user so that we can come up with an approximate list of friends. For this, we analyze the interactions of other users with the user. For example, if a person makes a comment about a post shared by the user, then we consider this person as being a friend of the user. So, this list includes more users than the actual list of friends of the user. Consider the user N_3 in Table 5. The actual number of friends for this user is 671. However, by analyzing the interaction data of the user, we come up with a list of 1060 users. Since the constructed list is only a partial view of the social network, our tool may not detect all of the violations. Moreover, the approximate list of friends may contain users who are not actual friends of the user (e.g., a friend of friend of the user will be included in the approximate list as a result of liking a post of the user).

[4] `https://www.mongodb.com/`

In such cases, the tool can report false positive violations. For example, if the user does not want her content to be seen by her friends, the tool can report a violation where a friend of friend of the user sees her content. However, if PriGuardTool was a service of the online social network with access to more information, such false positives would not take place.

Ontology Generation: Recall that PriGuardTool makes use of ontologies to keep information about the social network domain and the user. The user data, which is a JSON document, should be transformed into class and property assertions in PriGuard ontology. This transformation is realized by a Java application, which parses a JSON document and generate an ontology for the user. We use Apache Jena[5], which is an open-source Java framework to work with ontologies. The user may choose to check for privacy violations for a subset of her posts. Hence, ontologies of different sizes can be generated per request.

Note that the ontology generation module can take a long time if the user has lots of friends and posts. Hence, we adopt multi-threading to generate large ontologies. It is important to keep large ontologies in a database since privacy violations can also be detected offline. The maximum size of document that can be stored in MongoDB is 16MB. We use GridFS specification in MongoDB, which divides a document into various chunks that are stored separately as documents.

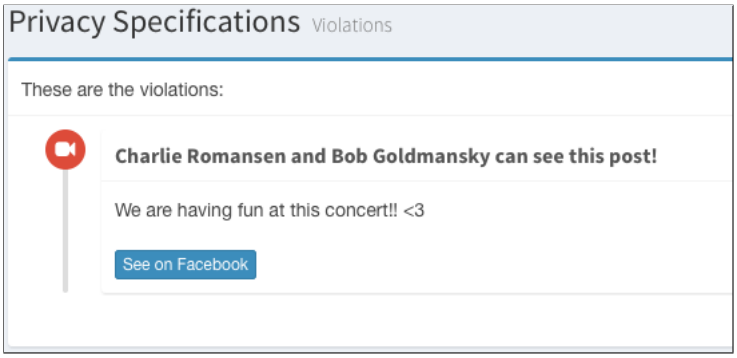

Fig. 3. Alice checks the posts that violate her privacy.

Detection Results: The users input their privacy concerns to detect privacy violations on Facebook as shown in Figure 1. Once the user checks for violations, a list of posts that violate the privacy of the user are displayed on the Web application. For example, Alice did not want Bob and Charlie to see her medium posts. When she checks for violations, she is notified that Charlie's post violates her privacy as shown in Figure 3. Here, Alice can get in touch with Charlie so that he modifies or removes this post since she is not the owner that post.

[5] `https://jena.apache.org/`

PriGuardTool can be used in two modes: online and offline. In both modes, agents use the user data to generate an ontology, which is loaded into memory for checking privacy violations. In online mode, PriGuardTool only considers posts that have been shared about the user in last three months. We do this to return recent privacy violations first in a short time. However, in offline mode, privacy violations are detected by the use of large ontologies. The user can also check the detection results that have been computed in offline mode. Then, the user can try to minimize the privacy violations to occur by modifying the posts if possible.

5 Evaluation

In the context of privacy, it is difficult to evaluate approaches and tools since there are no established data sets. Moreover, privacy is subjective hence it becomes difficult to talk of a gold standard that works for all. One way to go about this is to create synthetic data. However, ensuring that the synthetic data will adhere to real life properties is also difficult. Instead of working with synthetic data, it is ideal to work with real users. For this, we show the applicability of PriGuard approach in a Web application that is integrated to Facebook.

5.1 Experiments with Facebook Users

To evaluate our PriGuardTool implementation, we have worked with real data of Facebook users. We have collected data from Facebook users who used our tool to protect their privacy. Here, we generate five ontologies regarding the user data. The first four ontologies include posts shared last one month, three months, six months and last year. The fifth ontology includes the latest five hundred posts shared by the user. Additionally, the users specified their privacy concerns, which were translated into commitments. Then, the user agents checked for commitment violations in generated ontologies to report privacy violations.

We perform our experiments on Intel Xeon 3050 machine with 2.13 GHz and 4 GB of memory running Ubuntu 14.04 (64-bit). In Table 5, we present the evaluation results for three Facebook users. Each user inputs a privacy concern such that she chooses five people who should not see her medium posts. Then, the user checks for privacy violations. The user agent transforms this privacy concern into a commitment. Then, the user agent searches for commitment violations and reports if any.

The users have different numbers of friends and posts (N_1, N_2 and N_3). For each generated ontology of the user, we give information about the number of posts and the number of detected violations. Moreover, we measure the time that it takes to detect violations and to generate the corresponding ontology. For example, the user N_2 has 590 friends and 1894 posts. Her ontology includes information about posts shared in last six months. This ontology was generated in 10.79 seconds from the 51 posts she has made on Facebook. The tool detected

Table 5. Results for Facebook Users

$N_x(Friend\#, TotalPost\#)$		1mo.	3mo.	6mo.	12mo.	All
$N_1(293, 123)$	Post Number	2	9	27	47	123
	Violation Number	1	8	25	43	100
	Detection Time (s)	0.65	1.21	5.5	11.36	26.08
	Ontology Gen. Time (s)	1.2	2.24	4.6	6.34	11.12
$N_2(590, 1894)$	Post Number	5	19	51	134	500
	Violation Number	5	14	37	89	332
	Detection Time (s)	3.07	5.16	18.48	70.87	696.51
	Ontology Gen. Time (s)	2.33	6.51	10.79	18.07	33.7
$N_3(1060, 2945)$	Post Number	18	77	124	330	500
	Violation Number	9	44	69	164	237
	Detection Time (s)	3.28	76.74	187.53	783.06	1285.73
	Ontology Gen. Time (s)	3.34	9.85	16.23	41.23	67.14

37 privacy violations regarding the user's privacy concerns. The detection took 18.48 seconds. Whenever the social network of a user is small in size, the time for generating an ontology and detecting violations is less. For example, it takes only 11.12 seconds to generate an ontology for N_1 and 26.08 seconds for detecting 100 violations when we consider all posts. However, it takes longer when users are part of a large network. Even if the ontology generation time is reasonable (i.e., 67.14 seconds to generate the largest ontology for N_3), the detection takes a long time since the axiom number in the ontology increases as the result of ontological reasoning. For example, for N_3, the detection took approximately 20 minutes. Hence, such a detection should be done in offline mode if the detection is not achieved in a distributed manner as we do here. In online mode, the tool can report results in less than 80 seconds (considering that the user N_3 is a very active user) since we only consider posts shared in last three months. The user can then check the privacy violations and try to minimize them. She can modify the post attributes if she is the owner of the violating post. Otherwise, she can contact the post's owner to modify that post or to remove it completely.

5.2 Variations on Example Scenarios

We introduce two more examples that demonstrate privacy violations in an online social network. The first example requires multiple posts to be processed together to identify a privacy violation. We show that PriGuardTool can detect this successfully. The second example contains a privacy violation that can only be detected by processing non-structured data about the post (e.g., the image or text). In its current form, PriGuardTool cannot accommodate such processing and thus cannot detect the violation.

Consider the following example that shows how a privacy violation occurs indirectly in the presence of other users' posts.

Example 2 Bob shares a picture where he tags Diane. After a while Diane shares her location in a post. Alice and Charlie, who are friends of Bob, get to know Bob's location. However, Bob did not want to reveal his location.

In Example 2, a privacy violation occurs through inference. By combining Bob's post with Diane's post, one can infer Bob's location (see the inference rule r_8). However, in order to detect such violations, we should be able to collect Diane's posts as well. In the current implementation, we focus on collecting the user's data. For this example, Diane's post would not be extracted since it does not have any explicit tag for Bob. Note that PriGuardTool is able to detect violations of different types by the use of semantic rules when data is available. Another solution would be to integrate PriGuardTool to Facebook. In another words, if PriGuardTool ran as an internal application rather than an external one, then it would have access to the data and detect the privacy violation easily.

In the following example, the user shares a post that includes textual information, which reveals the location of the user.

Example 3 Bob shares a status message: "Hello Las Vegas, nice to finally meet you!". This message is shared with his friends.

In Example 3, Bob discloses his location himself. Hence, a privacy breach occurs because of the user itself. However, such a privacy violation cannot be identified by PriGuardTool because current agents do not analyze textual information to extract meaningful information. That is, a human can easily understand that Las Vegas is a city and that Bob is currently there. However, an agent would need to use Natural Language Processing (NLP) tools to find that Las Vegas is a location name, and the post being shared is indeed a location post. Thus, his friends reading this message would be violating Bob's privacy. This task is not straightforward in the context of privacy. An agent can recognize entities in a text by the use of external tools. However, it is unknown how these entities would affect the privacy of the user. We leave this point as a future work.

6 Discussion

This paper describes PriGuardTool, which is a concrete implementation of PriGuard, a semantic approach for detecting privacy violations in OSNs. PriGuardTool allows a user to specify her privacy constraints using a Web-based interface. The specified constraints are then converted into commitments. The tool then checks for commitment violations in a given system, which signals a privacy breach.

6.1 Related Work

While various approaches for privacy management exist, the number of tools is scarce. CoPE is a collaborative privacy management system that is developed to run as a Facebook application [24]. The idea is that each post is co-owned by multiple users that are affected by the post; e.g., because the individual is tagged or mentioned in the post. First, each co-owner specifies her own privacy requirement on a particular post. Then, the co-owners vote on the final privacy requirement on the post. The post is shared accordingly.

FaceBlock is an application designed to preserve the privacy of users that use Google Glass [20]. Given that interactions happen more seamlessly with wearable devices, it is possible that an individual takes a picture in an environment and shares it without getting explicit consent from others in the environment. To help users manage their privacy, FaceBlock allows users define their privacy rules with Semantic Web Rule Language (SWRL) and uses a reasoner to check whether any privacy rule is triggered. If so, FaceBlock obscures the face of the user before sharing the picture.

PriNego [17,9] and PriArg [11] are systems that have been built over the same framework. Users' privacy constraints are represented with SWRL rules. PriNego is a negotiation framework that allows users to negotiate their privacy constraints before a post is shared. At each iteration of the negotiation, a given post is updated based on privacy concerns of the users. For example, after the negotiation is done, the post might have fewer members in its audience list or fewer individuals tagged. PriArg uses argumentation to facilitate agreement among users. It enables users to attack each other's privacy concerns with information they provide (i.e., arguments). At the end of the argumentation, whether a post will be shared or not is decided.

Kafali *et al.* develop PROTOSS [8], where the privacy agreements are again represented with commitments. However, in that work, the commitments are taken from the user as opposed to being generated as we have done here. Further, the system evolution is being tested for violations rather than the current state of the system. PROTOSS uses model checking hence in a given state of the social network, all the possible states are generated. In PriGuardTool, we are only concerned with a single state of the system, we can detect violations much faster and with less memory requirements then them.

Akcora, Carminati and Ferrari develop a graph-based approach and a risk model to learn risk labels of strangers; e.g., friends of friends [1]. The intuition is that these will enable them to detect individuals who are likely to violate privacy constraints. Our focus is not on identifying potential individuals that can view private data but on detecting violations through interactions on the OSN.

Liu and Terzi address the privacy problem in OSNs from the user's perspective [16]. They propose a model to compute a privacy score of a user. The privacy score increases with the *sensitivity* and *visibility* of the revealed information. *Sensitivity* is specific to a profile item while *visibility* of a profile item depends on the privacy settings of the user. It would be interesting to capture these concepts in PriGuard ontology and make inferences based on that.

Squicciarini *et al.* propose PriMa (Privacy Manager), which supports semi-automated generation of access rules according to the user's privacy settings and the level of exposure of the user's profile [23]. They further provide quantitative measurements for privacy violations. Quantifying violations is an interesting direction that we want to investigate further. Our use of an ontology can make it possible to infer the extents of the privacy violation, indicating its severity.

Fang and LeFevre propose a privacy wizard that automatically configures the user's privacy settings based on an active learning paradigm [4]. Their approach is based on the user's privacy preferences while we consider the privacy preferences of the user and her social graph. Moreover, we focus on detecting privacy violations that would happen because of conflicting privacy concerns of the users.

Krishnamurthy points out the need for privacy solutions to protect the user data from all entities who may access it [14]. He suggests that OSN users should know what happens to their privacy as a result of their actions. For this, a Facebook extension called Privacy IQ is developed where users can see the privacy reach of their posts and the effect of their past privacy settings. PriGuardTool is similar to this work in that we can also compare the user's privacy expectations with the actual state of the system. However, our major contribution is on detecting privacy breaches that take place because of interactions among users and inferences on information.

6.2 Limitations and Future Developments

The main obstacle we faced in adapting PriGuardTool to Facebook was that the current Facebook API does not allow a user to obtain much of the information she sees programatically. For example, a user can see her list of friends when she logs in to Facebook, but she cannot get the same list using the API. Hence, we could only construct a partial list of friends using information such as comments, tags, and so on. Although most of the time, the constructed information was sufficiently accurate, it would have been much easier if the agent could access the information to begin with.

In this work, we assume that users are able to input their privacy concerns in a fine grained way. However, users have difficulties to specify their privacy concerns even if they have the necessary tools [4]. To solve this problem, one approach would be to conduct user studies to understand the user needs better. As a result, we can design better user interfaces that guide the users in specifying their privacy expectations. Another approach would be to learn the privacy concerns of the user automatically [19,10]. This would minimize the user burden and errors by suggesting privacy configurations.

The current system supports commitments between a user and the online social network. However, in principle, if the online social network itself supports a distributed architecture (e.g., GnuSocial[6]), then individual users will be responsible for managing their content and thus the system would have to support

[6] `https://gnu.io/social/`

commitments among users. This would lead to interesting scenarios and could serve as a natural domain to demonstrate operations on commitments. For example, Bob could commit to Alice not to share her pictures and then follow up with his friends to ensure that Alice's pictures are not shared. This could lead to multiple commitments being merged and manipulated to preserve privacy and give rise to composition of commitments for representing realistic scenarios [2].

Another important improvement could be to detect privacy violations in a distributed manner. The current implementation receives a state of the system and checks for possible violations in that state. A distributed implementation could help process the state considerably faster. This would enable the tool to be used online easily.

Acknowledgments

This work is supported by TUBITAK under grant 113E543. This work extends the demonstration paper that was presented at AAMAS 2016 [13]. We thank Hamza Ozturk and Safa Orhan for helping with integrating PriGuardTool to Facebook.

References

1. Akcora, C.G., Carminati, B., Ferrari, E.: Risks of friendships on social networks. In: IEEE International Conference on Data Mining (ICDM). pp. 810–815 (2012)
2. Baldoni, M., Baroglio, C., Chopra, A.K., Singh, M.P.: Composing and verifying commitment-based multiagent protocols. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI). pp. 10–17 (2015)
3. Carminati, B., Ferrari, E., Heatherly, R., Kantarcioglu, M., Thuraisingham, B.: Semantic web-based social network access control. Computers & Security 30(2), 108–115 (2011)
4. Fang, L., LeFevre, K.: Privacy wizards for social networking sites. In: Proceedings of the 19th international conference on World wide web. pp. 351–360. ACM (2010)
5. Gürses, F., Berendt, B.: The social web and privacy: Practices, reciprocity and conflict detection in social networks. Privacy-Aware Knowledge Discovery. Chapman & Hall/CRC Press, New York, NY pp. 395–432 (2010)
6. Heussner, K.M.: Celebrities' photos, videos may reveal location. ABC News, Available at: `http://goo.gl/sJIFg4`
7. Hu, H., Ahn, G.J., Jorgensen, J.: Multiparty access control for online social networks: model and mechanisms. IEEE Transactions on Knowledge and Data Engineering 25(7), 1614–1627 (2013)
8. Kafalı, O., Günay, A., Yolum, P.: Detecting and predicting privacy violations in online social networks. Distributed and Parallel Databases 32(1), 161–190 (2014)
9. Keküllüoğlu, D., Kökciyan, N., Yolum, P.: Strategies for privacy negotiation in online social networks. In: Proceedings of the 1st International Workshop on AI for Privacy and Security (PrAISe). pp. 2:1–2:8 (2016)
10. Kepez, B., Yolum, P.: Learning privacy rules cooperatively in online social networks. In: Proceedings of the 1st International Workshop on AI for Privacy and Security (PrAISe). pp. 3:1–3:4. ACM (2016)

11. Kökciyan, N., Yaglikci, N., Yolum, P.: Argumentation for resolving privacy disputes in online social networks: (extended abstract). In: Proceedings of the 15th International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016. pp. 1361–1362 (2016)
12. Kökciyan, N., Yolum, P.: Priguard: A semantic approach to detect privacy violations in online social networks. IEEE Transactions on Knowledge and Data Engineering 28(10), 2724–2737 (2016)
13. Kökciyan, N., Yolum, P.: PriGuardTool: A tool for monitoring privacy violations in online social networks (demonstration). In: Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 1496–1497 (2016)
14. Krishnamurthy, B.: Privacy and online social networks: can colorless green ideas sleep furiously? IEEE Security and Privacy 11(3), 14–20 (May 2013)
15. Lampinen, A., Lehtinen, V., Lehmuskallio, A., Tamminen, S.: We're in it together: interpersonal management of disclosure in social network services. In: Proceedings of the SIGCHI conference on human factors in computing systems. pp. 3217–3226. ACM (2011)
16. Liu, K., Terzi, E.: A framework for computing the privacy scores of users in online social networks. ACM Transactions on Knowledge Discovery from Data (TKDD) 5(1), 6:1–6:30 (2010)
17. Mester, Y., Kökciyan, N., Yolum, P.: Negotiating privacy constraints in online social networks. In: Koch, F., Guttmann, C., Busquets, D. (eds.) Advances in Social Computing and Multiagent Systems, Communications in Computer and Information Science, vol. 541, pp. 112–129. Springer International Publishing (2015)
18. Motik, B., Patel-Schneider, P.F., Parsia, B., Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., et al.: Owl 2 web ontology language: Structural specification and functional-style syntax. W3C recommendation 27(65), 159 (2009)
19. Mugan, J., Sharma, T., Sadeh, N.: Understandable learning of privacy preferences through default personas and suggestions. Tech. Rep. CMU-ISR-11-112, Carnegie Mellon University, School of Computer Science (2011)
20. Pappachan, P., Yus, R., Das, P.K., Finin, T., Mena, E., Joshi, A.: A semantic context-aware privacy model for faceblock. In: Proceedings of the 2nd International Conference on Society, Privacy and the Semantic Web - Policy and Technology. pp. 64–72. PrivOn (2014)
21. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Transactions on Database Systems 34(3), 16 (2009)
22. Singh, M.P.: An ontology for commitments in multiagent systems. Artificial Intelligence and Law 7(1), 97–113 (1999)
23. Squicciarini, A.C., Paci, F., Sundareswaran, S.: PriMa: a comprehensive approach to privacy protection in social network sites. Annals of Telecommunications/Annales des Télécommunications pp. 1–16 (2013)
24. Squicciarini, A.C., Xu, H., Zhang, X.L.: Cope: Enabling collaborative privacy management in online social networks. Journal of the American Society for Information Science and Technology 62(3), 521–534 (2011)
25. Such, J.M., Rovatsos, M.: Privacy policy negotiation in social media. ACM Transactions on Autonomous and Adaptive Systems (TAAS) 11(1), 4:1–4:29 (2016)
26. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: applying event calculus planning using commitments. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 527–534 (2002)