A View-Based Access Control Model for SPARQL

Alban Gabillon, Léo Letouzey

To cite this version:

Alban Gabillon, Léo Letouzey. A View-Based Access Control Model for SPARQL. 4th International Conference on Network and System Security (NSS), 2010, Sep 2010, Melbourne, Australia. pp.105 - 112, ⟨10.1109/NSS.2010.35⟩. ⟨hal-01020253⟩

HAL Id: hal-01020253

https://hal.science/hal-01020253v1

Submitted on 11 Jul 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL Authorization

A View Based Access Control Model for SPARQL

Alban Gabillon, Léo Letouzey
Université de la Polynésie Française
BP 6570, 98702 FAA'A
French Polynesia
{alban.gabillon,leo.letouzey}@upf.pf

Abstract

Existing security models for RDF use RDF patterns for defining the security policy. This approach leads to a number of security rules which rapidly tends to be unmanageable. In this paper we define a new security model which follows the traditional approach of creating security views, which has long been used by SQL database administrators. Our model first logically distributes RDF data into SPARQL views and then it defines security rules regulating SPARQL access to views. Moreover our model supports rights delegation and dynamic security rules (i.e. rules which can be active or not, depending on the context).

1. Introduction

Several access control models for RDF (Resource Description Framework) [15] data have been proposed [2][3][4][5]. Most of these models have the same two drawbacks: (1) the security policy consists of RDF patterns defining the RDF triples which can be accessed. Even though, this approach allows for fine grained access control, it does not scale to large RDF datasets since the number of security rules becomes rapidly excessive; (2) none of these models include an administration model specifying how the security policy can be updated.

The SQL (Structured Query Language) security model is a View Based Access Control model (VBAC) for relational databases which has proved to be practical and scalable. In this model, each application designer *owns* a set of SQL tables for which she manages the security policy. Basically, to define access rights, the application designer proceeds as follows: (1) she first defines a set of SQL *views*. A view is a virtual table that consists of columns and rows from one or more tables. Concretely, a view is *a query stored as an object* that derives its data from one or more tables. A view can be referenced in a query like any table. If a user query referencing a view is submitted to the SQL engine then the query defining the view is first dynamically evaluated; (2) then she grants access rights on views (and possibly tables) to users and/or roles. In SQL, the existing access rights correspond to the four SQL query forms namely, SELECT, INSERT, DELETE and UPDATE. By managing access rights on views rather than on tables, the application designer has more flexibility to restrict access to rows and columns of data. Views provide also an elegant way of implementing security rules involving data distributed into several tables. SPARQL (recursive acronym that stands for SPARQL Protocol and RDF Query Language) [12] is a standardized query language for RDF data. SPARQL queries reference one or several RDF *graphs*. SPARQL has four query forms (SELECT, CONSTRUCT, ASK, DESCRIBE). Both the CONSTRUCT and DESCRIBE queries return an RDF graph. The core of the security model we propose in this paper is basically an interpretation of the SQL security models for SPARQL where (i) RDF graphs play the role of tables, (ii) RDF views are CONSTRUCT or DESCRIBE queries stored as objects and (iii) the security rules regulate the execution of the four SPARQL query forms. On top of this, our model supports rights delegation and enables dynamic security rules, i.e. rules which become active only if a certain context is true. Organization of the remainder of this paper is the following. In section 2, we quickly introduce SPARQL. In section 3, we review existing access control models for RDF data and we give an example motivating our work. Section 4 describes our security model. In section 5, we sketch the architecture of a secure proxy for RDF data implementing our model. In particular we give the algorithms used by the Policy Decision Point and the Policy Enforcement Point. Finally, section 6 concludes this paper.

2. SPARQL

With the greater adoption of RDF, many languages have been proposed to query RDF repositories (RDQL (RDF

Data Query Language) [10], RQL (RDF Query Language) [11], SPARQL [12]). Since January 2008, SPARQL is the W3C recommended language to query RDF document. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions.

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd="http://www.w3c.org/2001/XMLSchema#">
	<foaf:Person>
		<foaf:name>Bob</foaf:name>
		<foaf:surname>Rob</foaf:surname>
		<foaf:interest>IT</foaf:interest>
		<foaf:img>portrait.jpg</foaf:img>
		<foaf:mbox>rob@home.com</foaf:mbox>
		<foaf:gender>Male</foaf:gender>
		<foaf:knows>
			<foaf:Person>
				<foaf:name>Alice</foaf:name>
				<foaf:mbox>alice@home.com</foaf:mbox>
				<foaf:surname>Ali</foaf:surname>
				<foaf:interest>studies</foaf:interest>
				<foaf:interest>Maths</foaf:interest>
				<foaf:age rdf:datatype="xsd:integer">22</foaf:age>
				<foaf:based_near>Paris</foaf:based_near>
			</foaf:Person>
		</foaf:knows>
		<foaf:knows>
			<foaf:Person>
				<foaf:name>Hans</foaf:name>
				<foaf:mbox>Hans@home.com</foaf:mbox>
				<foaf:interest>IT</foaf:interest>
				<foaf:interest>Maths</foaf:interest>
				<foaf:based_near>Paris</foaf:based_near>
			</foaf:Person>
		</foaf:knows>
		<foaf:knows>
			<foaf:Person>
				<foaf:name>Charlie</foaf:name>
				<foaf:age rdf:datatype="xsd:integer">20</foaf:age>
				<foaf:mbox>Charlie@home.com</foaf:mbox>
				<foaf:based_near>Papeete</foaf:based_near>
				<foaf:interest>Maths</foaf:interest>
				<foaf:interest>Diving</foaf:interest>
			</foaf:Person>
		</foaf:knows>
	</foaf:Person>
</rdf:RDF>
```

Figure 1 myfoaffile.rdf

A SPARQL query is of one of the 4 following types: SELECT, ASK, CONSTRUCT and DESCRIBE. SELECT queries are the most common. A SELECT query returns all, or a subset of, the variables bound in a query pattern match (see example below). A ASK query returns True or False depending on whether a graph pattern matches or not. The two other query types return RDF graphs. A CONSTRUCT query returns a RDF graph constructed by substituting variables in a set of triple templates (see section 3 for an example of a CONSTRUCT query). A DESCRIBE form returns a single result RDF graph containing RDF data about resources.

In order to describe how SPARQL queries are processed against RDF repositories, let us consider the RDF document in figure 1. This document is defined in the FOAF (Friend Of A Friend) ontology [19]. The FOAF project is a community driven effort to define an RDF vocabulary for expressing metadata about people, and their interests, relationships and activities.

Let us consider the following SELECT query:

```
  PREFIX foaf:<http://xmlns.com/foaf/0.1/>
1.SELECT ?name
  WHERE {
2.?x foaf:name "Bob" .
3.?x foaf:knows ?y .
4.?y foaf:name ?name .
  }
```

Variable `x` line 2 is bound to subject foaf:Person whose predicate foaf:name targets object “Bob”. Variable `y` (line 3) is bound to the foaf:Person elements that are known by `x`. In line 4, variable `name` matches objects `o` so that triple `(s, p, o)` belongs to the RDF document, with `s` being one of the values found for `y` and `p` being predicate foaf:name. Answer to this query is the set {Alice, Hans and Charlie}.

The result set of a SPARQL SELECT (or ASK) query can be serialized to XML. The SPARQL Variable Binding Results XML Format is the W3C recommended language for serializing the result of SPARQL query to a SPARQL Results Document (SRD) [20].

3. Related Works and Motivations

In this section, we review the main existing access control models for RDF data [2][3][4][5]. Several authors (see [2][3] for instance) have underlined the fact that existing access control models for XML data cannot be applied to RDF data. We agree on this point. Therefore, we shall not consider access control models for XML in this related work section.

In all existing access control models for RDF, security rules use RDF patterns to match RDF triples. An RDF pattern is an RDF triple `(subject, predicate, object)` where `subject`, `predicate` and `object` can be substituted by variables. In [2], the authors define a set of actions that can be performed on an RDF store. They define several operations for updating the store and two operations for querying the store. The security policy consists of permissions or prohibitions to perform actions on some RDF triples. Each permission or prohibition can be subject to a condition. This condition is either based on metadata that the RDF store maintains or on the triples themselves. Enforcement of the security policy (for query actions) requires filtering out unauthorized triples from

the result set. In [3], the authors deal with multilevel security in RDF stores. They assign security labels to RDF triples. In order to prevent unauthorized inferences, they consider the entailment rules defined in the W3C RDF Semantics [17] and suggest some rules for automatically assigning security labels to entailed RDF statements. In [4], the authors consider the read access only. The security policy consists of permissions or prohibitions to access some RDF statements. Each authorization can be subject to a condition based on contextual information. Regarding policy enforcement, authors argue that approaches used in [2] and [3] are not efficient since they require to instantiate the graph patterns used in the security policy. They propose instead an algorithm to rewrite a given query into a secure query according to the security policy. In [5], the authors consider the read access only. Like in [16], an authorization (positive or negative) can be recursive or local. If it is recursive then it propagates (explicit propagation) to lower classes and lower properties based on the RDF schema. The authors also define the concept of implicit authorization in the RDF inference. The authors propose then a solution to detect the conflicts which may arise between authorizations.

Most of these models have the same major drawback: they use RDF patterns for identifying the RDF triples which can be accessed. This approach does not scale to large RDF datasets. As a matter of fact, let us consider the FOAF rdf file myfoaffile.rdf shown in figure 1 (section 2) and consider the following security policy applying to this file:

"Alice is permitted to see name, surname, email and interests of Bob's friends who live in Paris and who are interested in mathematics."

Figure 2 shows how to express this security policy in the framework of the model defined in [2]. Figure 3 shows the same policy in the framework of the model defined in [4].

As we can see both models requires writing 4 rules, one for each of the following patterns:

```
(X, foaf:name, Y)
(X, foaf:surname, Y)
(X, foaf:interest, Y)
(X, foaf:mbox, Y)
```

Moreover, since access to these patterns is subject to some conditions, these conditions have to be repeated in each rule. In fact, using RDF patterns for identifying RDF triples leads to a number of security rules which quickly tends to be unmanageable and unreadable.

Existing models have another drawback. None of them include an administration model specifying how the security policy can be updated. In fact, all existing models implicitly assume that the definition of the security policy should be carried out by a central authority. However, in an open environment like the Web, metadata come from different sources and should be managed in a decentralized way.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
Permit(see(Alice, (X, foaf:name, ?))) :-existTriple(Y, rdf:type, foaf:Person)
                                        & existTriple(Y, foaf:name, "Bob")
                                        & existTriple(Y, foaf:knows, X)
                                        & existTriple(X, foaf:interest, "Maths" )
                                        & existTriple(X, foaf:based_near, "Paris")

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
Permit(see(Alice, (X, foaf:mbox, ?))) :-existTriple(Y, rdf:type, foaf:Person)
                                        & existTriple(Y, foaf:name, "Bob")
                                        & existTriple(Y, foaf:knows, X)
                                        & existTriple(X, foaf:name, ?)
                                        & existTriple(X, foaf:interest, "Maths" )
                                        & existTriple(X, foaf:based_near, "Paris")

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
Permit(see(Alice, (X, foaf:interest, ?))) :-existTriple(Y, rdf:type, foaf:Person)
                                        & existTriple(Y, foaf:name, "Bob")
                                        & existTriple(Y, foaf:knows, X)
                                        & existTriple(X, foaf:name, ?)
                                        & existTriple(X, foaf:interest, "Maths" )
                                        & existTriple(X, foaf:based_near, "Paris")

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
Permit(see(Alice, (X, foaf:surname, ?))) :-existTriple(Y, rdf:type, foaf:Person)
                                        & existTriple(Y, foaf:name, "Bob")
                                        & existTriple(Y, foaf:knows, X)
                                        & existTriple(X, foaf:name, ?)
                                        & existTriple(X, foaf:interest, "Maths" )
                                        & existTriple(X, foaf:based_near, "Paris")
```

Figure 2. Policy Based Access Control for An RDF Store. Reddivari, Finin, Joshi 2005 [2]

The model we define in this paper does not have the inconveniences of the existing models. In our model, each RDF graph has an owner who manages the security policy protecting the graph. Typically, the owner of a graph first creates various views on her graph, by means of SPARQL CONSTRUCT or DESCRIBE queries. She then grants the SPARQL SELECT privilege on these views to other users. As a matter of fact, for writing a security policy saying that Alice has the permission to see name, surname, email and interests of Bob's friends who live in Paris and who are interested in mathematics, Bob, owner of the foaf file myfoaffile.rdf (figure 1), proceeds as follows: (1) he first creates the view shown in figure 4. This view definition is referred by the URL foafview.txt. It contains a CONSTRUCT statement which selects the name, surname, mbox and interests of Bob's friends who live in Paris and who are interested in mathematics (2) he then defines a single simple security rule granting the

SELECT privilege on the view foafview.txt to Alice (see section 3 for the definition of security rules):

```
Permit(Alice,SELECT,foafview.txt)
```

This rule says Alice is permitted to execute a SELECT statement on the view which is referred by the URL foafview.txt.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
Allow access to triples (X, foaf:name, N) IF
        Requester = 'Alice' AND
        (Y, rdf:type, foaf:Person) AND
        (Y, foaf:name, "Bob") AND
        (Y, foaf:knows, X) AND
        (X, foaf:interest, "Maths) AND
        (X, foaf:based_near, "Paris")

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
Allow access to triples (X, foaf:mail, M) IF
        Requester = 'Alice' AND
        (Y, rdf:type, foaf:Person) AND
        (Y, foaf:name, "Bob") AND
        (Y, foaf:knows, X) AND
        (X, foaf:name, N) AND
        (X, foaf:interest, "Maths) AND
        (X, foaf:based_near, "Paris")

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
Allow access to triples (X, foaf:interest, M) IF
        Requester = 'Alice' AND
        (Y, rdf:type, foaf:Person) AND
        (Y, foaf:name, "Bob") AND
        (Y, foaf:knows, X) AND
        (X, foaf:name, N) AND
        (X, foaf:interest, "Maths) AND
        (X, foaf:based_near, "Paris")

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
Allow access to triples (X, foaf:surname, M) IF
        Requester = 'Alice' AND
        (Y, rdf:type, foaf:Person) AND
        (Y, foaf:name, "Bob") AND
        (Y, foaf:knows, X) AND
        (X, foaf:name, N) AND
        (X, foaf:interest, "Maths) AND
        (X, foaf:based_near, "Paris")
```

Figure 3. Enabling Advanced and Context-Dependent Access Control in RDF Store. Abel et al, 2007 [4]

If Alice submits a SPARQL SELECT query on foafview.txt then, the CONSTRUCT query defining the view is first dynamically evaluated and then, the SELECT query submitted by Alice is evaluated on the RDF graph which is returned by the CONSTRUCT query.

The advantage of using security views for managing RDF data is obvious. Since a view represents a comprehensive set of semantically related data, we do not need to include these semantics relationships as conditions in the security rules, as it is done in models [2] and [4]. Moreover, since we can group several RDF patterns of interest into the same view, we can reduce the number of rules. Consequently, we gain in readability and concision. Again, we would like to reiterate that this approach has long been used successfully by many SQL database administrators.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
CONSTRUCT {
        ?x foaf:name ?n .
        ?x foaf:interest ?int .
        ?x foaf:mbox ?m .
        ?x foaf:surname ?sn .
        }
WHERE {
        ?y rdf:type foaf:Person .
        ?y foaf:name "Bob" .
        ?y foaf:knows ?x .
        ?x foaf:name ?n .
        ?x foaf:based_near "Paris" .
        ?x foaf:interest "Maths" .
        ?x foaf:interest ?int .
        OPTIONAL { ?x foaf:mbox ?m .}
        OPTIONAL { ?x foaf:surname ?sn .}
        }
```

Figure 4. View foafview.txt. Interest and mbox are defined as OPTIONAL in the query since some persons may not have surname and mail box

4. Security Model

Subjects, Objects, Actions, Views and Contexts

Definition of an access control model requires the definition of the objects to be protected, the actions to be executed on objects and the subjects that execute the actions [9]. In our model, subjects are users or processes. Objects are either RDF graphs or views. Views are RDF graphs created by means of CONSTRUCT or DESCRIBE queries. Actions (privileges) correspond to the four SPARQL query types, namely SELECT, ASK, CONSTRUCT and DESCRIBE. Security rules can be dynamic i.e. become active only if a certain context is true. As it is done in the ABAC model [1] or in the OrBAC model [18], we define a context as a conjunction and/or disjunctions of logical conditions applying to the subject, the object and the environment. Handling contexts allows us in particular to write security rules which do not require authenticating every user. This feature is particularly desirable in an open environment like Internet where it is often unrealistic to authenticate every user.

Security Policy Formulation

Definition of an access control model requires also the definition of a language for assigning rights to users. As in many Rule Based Access Control models [21][22], our

language is based on first-order logic. Syntax of an authorization rule is the following:

`condition → Permit(s,a,o)`

where `condition` is a possibly empty set of constraints applying to the subject, the object and the environment. `Permit(s,a,o)` reads "subject `s` is permitted to execute action `a` on object `o`". If a security rule is not constrained (i.e. condition is empty) then is it of the form `Permit(s,a,o)` (see example in section 3). The *default policy* of our model is *closed*. This means that, given a subject `s` requesting to execute action `a` on object `o`, if `Permit(s,a,o)` cannot be derived from the security policy then subject `s` should be denied to execute action `a` on object `o`. In the following examples, for better readability, we omit the universal quantifiers.

With the following statement, Bob grants privilege ASK on foafview.txt to everybody but only during daytime:

```
Time(CLOCK,t) ∧ (t > 8) ∧ (t < 20)
        → Permit(s,ASK,foafview.txt)
```

Predicate `Time(CLOCK,t)` reads "current time (given by the system CLOCK) is `t`"

With the following statement, Bob grants privilege SELECT on foafview.txt to whoever connects from the university network:

```
IP(s,i) ∧ NetUniv(i)
        → Permit(s,SELECT,foafview.txt)
```

Predicate `IP(s,i)` reads "IP address of `s` is `i`". Predicate `NetUniv(i)` reads "`i` belongs to the university network"

Some existing security models for RDF deal with *negative authorizations*. For example, the security policy in [4] uses allow/deny rules. Negative authorizations allow the security administrator to specify an exceptional prohibition to a general permission (or vice versa). The problem with having positive and negative authorizations is conflict management. There have been many research works on this issue mainly in the area of access control models for XML (see [16] for instance). However, these works should be revised to take into account the fact that in modern applications security rules are rather dynamic and are based on contextual conditions. Therefore, writing security policies including positive and negative authorizations requires being able to *predict* the potential conflicts which may arise between authorizations (see [23] for more details about this topic).

Administration

Most of existing models for RDF data are designed for centralized RDF stores. They implicitly assume that the definition of the security policy should be carried out by a central authority. On the contrary, we designed our model with in mind a decentralized system where people create their own RDF data and publish them through a secure proxy whose function is to regulate access to these various datasets according to the security policies defined by the users themselves. Let us consider Bob who needs to publish his foaf RDF data and who needs to regulate access to these data. In our scenario, Bob will create several views on his original dataset. He will then publish his original dataset and his views through a secure proxy along with the security policy regulating access to the original dataset and the views. Typically, he will forbid everybody to directly access to the original dataset but he will define some rules regulating access to the views. The proxy will then be in charge of implementing the security policy. For defining the security policy, Bob uses the logical language defined in the previous subsection. In order to better structure the security policy, he may also create roles which will be local to his proxy schema. More formally, the administration model of our security model can be described as follows: each RDF graph/view has an owner who is the user who created it. The owner of a graph/view holds all privileges on it. Each user defines the security policy for the RDF graphs/views she owns by means of, (1) CONSTRUCT or DESCRIBE queries stored as objects, (2) a (possibly empty) role hierarchy and (3) logical security rules, as defined in the previous subsection. Creating roles and assigning roles to users or other roles can be done with the following two predicates:

`Role(r)` which reads "`r` is a role"
`Isa(r,r')` If `r` is a role then it reads "`r` is a sub-role of role `r'`". If `r` is a user then it reads "user `r` plays role `r'`".

Note that, regarding the role/user hierarchy, the following entailment rule applies:

`Isa(r,r') ∧ Isa(r',r") → Isa(r,r")`

For example, with the following logical facts, Bob creates role Friend which he grants to Alice:

```
Role(Friend)
Isa(Alice,Friend)
```

Roles are local to the schema of the user who created them. This means in particular that two different users may create a role Friend. This point deserves to be stressed since in many SQL databases (see Oracle [24] for example), roles are global to the system and therefore two different users cannot create two roles having the same name. Following principles of the ABAC model, we treat roles like any other subject attribute. Consequently, privileges are not granted to roles but to users who are member of roles. For example, for specifying that all his friends have the permission to create views on the view foafview.txt, Bob would write the following security rule:

```
Isa(s,Friend)
   → Permit(s,CONSTRUCT,foafview.txt)
```

Now, Bob's friends have the right to define views (i.e. CONSTRUCT queries stored as objects) on foafview.txt.

This latter security rule allows us to introduce the concept of *delegation*. We say that there is delegation when a user defines the security policy for data she does not own. In our model, this happens in the following case: if a user has been granted the CONSTRUCT privilege on a RDF dataset for which she is not the owner, then this user has the privilege to create views on this RDF dataset. Consequently, she can define the security policy regulating access to these views since she owns them. By regulating access to her views, she, in fact, indirectly regulates access to some source RDF data. As a matter of fact, let us consider the previous example where Bob grants to his friends the privilege CONSTRUCT on foafview.txt. Bob's friends are now *de facto* administrators of the RDF data included in the view foafview.txt, i.e. they can create views on foafview.txt and define the security policy protecting these views. In this example, Bob has *partially* delegated the administration of his original dataset to his friends since he has granted the CONSTRUCT privilege on foafview.txt and not on the original dataset myfoaffile.rdf. To summarise how delegation works in our model, we state the following principles: let `s` be the owner of an RDF dataset `r`.

- If `s` does not need to delegate the security administration of `r` then she shall not grant to anybody neither the CONSTRUCT nor the DESCRIBE privilege on `r` or on any view she would have created on `r`.
- If `s` needs to delegate the security administration of the whole set `r`, then she shall grant to another user the CONSTRUCT and DESCRIBE privileges on `r`.
- If `s` needs to delegate the security administration of a subset of `r`, then she would create a view computing this subset and grant to another user the CONSTRUCT and/or DESCRIBE privilege on that view.

Of course, our model enables *delegation chains* to be created. For example, if a friend of Bob creates a view `v` on foafview.txt and then grants to another user the CONSTRUCT privilege on view `v` then this other user will have the possibility to define views on `v` and consequently will have to define the security policy protecting these views.

5. Secure Proxy Server

Architecture

We have implemented our model within the framework of a proxy server. This proxy is online at the following URL:
`http://projets.upf.pf:8080/ProxyServer`
`login: guest        Password: GuestPass`

Figure 5 sketches the architecture of our proxy. Basically, our proxy uses three databases. The *proxy database* stores user account data. Users who are willing to publish data do need an account. End-users, who only need to query data may or not create an account, depending on whether they need to be authenticated or not to access a given resource. The *security policy database* contains the role hierarchy and the access control lists of each database schema. The *RDF database* stores RDF graphs and view definitions which are distributed into user database schemas. It is important to note that users always create graphs and views in their own schema. Therefore a user who has been granted a CONSTRUCT (or DESCRIBE) privilege on some data owned by another user, does need an account to create views from these data. For performance issue, our proxy manages a cache which stores the most frequently computed views. Our proxy is implemented as a Web application running on top of the Tomcat Application Server [13]. We use SESAME Java API [14] for storing and querying RDF data.

Access Control Lists (ACL)

Security Rules are implemented as ACLs. Each object (RDF dataset of view) is linked to an ACL. Figure 6 shows the ACL of foafview.txt corresponding to the rules defined in section 3 and 4. Whenever a user requests access to the view defined in foafview.txt, conditions applying to the requested privilege are evaluated. Access is granted if at least one of the conditions holds for the user requesting the access.

Policy Decision and Enforcement Points

The Internet Engineering Task Force (IETF) defines an abstract model for policy enforcement which is used in most commercial implementation of access control mechanisms. This abstract model makes a clear distinction between the ***P****olicy* ***D****ecision* ***P****oint* (PDP) component and the ***P****olicy* ***E****nforcement* ***P****oint* (PEP):

- The PEP intercepts the access request and forwards it to the PDP. After it has received the decision from the PDP, it enforces the decision against the requester.
- The PDP analyzes the access request, evaluates contextual conditions and then decides on the concrete outcome of the request (i.e. access granted or access denied).

The access control processor of our proxy server uses the following algorithm: let `u` be the user submitting query `q`. Let `s(q)` denote the source views/datasets queried by query `q`. Let `q(v)` denote the CONSTRUCT or DESCRIBE query defining view `v`. Let `o(v)` denote the owner of view `v`. Let `t(q)` denote the type (SELECT, ASK, CONSTRUCT or DESCRIBE) of query `q`.

```
If decision(u,q) then
        output(enforce(q))
Else
        output("Access Denied")
```

The PDP of our server computes function `decision` which is recursive. It takes as input a user and a query. It returns a boolean.

```
Function decision(u,q) : Boolean
for v in s(q)
     If Permit(u,t(q),v) cannot be derived from
      the Security Policy then
           Return False
     Else
           If v is a view
               q' ← q(v)
               u' ← o(v)
               If not decision(u',q')then
                    return False
return True
```

Regarding this algorithm, the following points should be noted:

- Facts belonging to the `Permit` predicate are derived after context evaluation.
- Variable `u` is not always instantiated when function `decision` is called since users may connect to the proxy without being authenticated (in that case access decisions are taken based on contextual conditions applying to them).
- If a user is permitted to query a view then this query may still be rejected if the owner of the view is forbidden to dynamically evaluate the view (recall that permissions are not always active since they are context-based).
- A view can be computed from several sources. If the security check fails for one of these sources then access is denied.

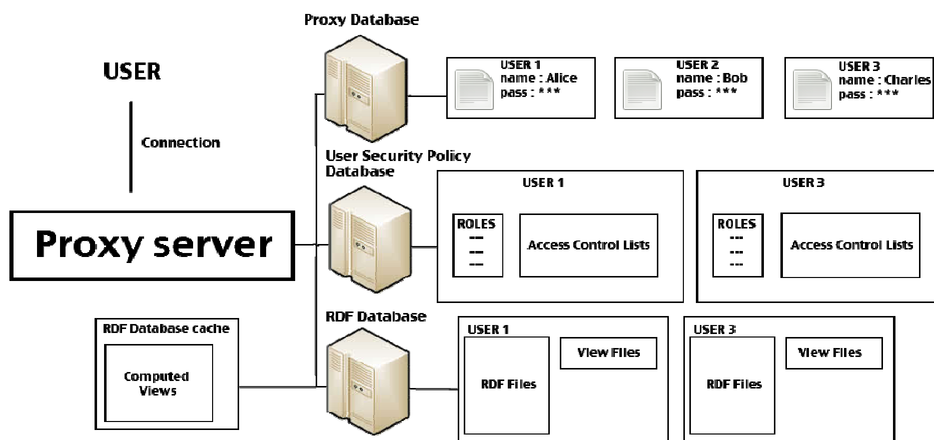

Figure 5. View Proxy Server

```
<?xml version="1.0"?>
<rdf:RDF xmlns:acl="http://home.org/acl#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
roles
        <acl:Role rdf:nodeID="ro_Client">
                <acl:Name>Client</acl:Name>
        </acl:Role>
        <acl:Role rdf:nodeID="ro_Friend">
                <acl:Name>Friend</acl:Name>
        </acl:Role>

RDF Data and Views
        <acl:RDFData rdf:nodeID="rd_myfoaffile">
                <acl:Name>myfoaffile</acl:Name>
                <acl:File>myfoaffile.rdf</acl:File>
        </acl:RDFData>

        <acl:View rdf:nodeID="v_foafview">
                <acl:Name>foafview</acl:Name>
                <acl:OnData>myfoaffile</acl:OnData>
                <acl:File>foafview.txt</acl:File>
        </acl:View>

ACL
<acl:Rule>
        <acl:Type>GrantRole</acl:Type>
        <acl:Role rdf:nodeID="ro_Friend"/>
        <acl:To>
                <acl:User>
                        <acl:Name>Alice</acl:Name
                </acl:User>
        </acl:To>
</acl:Rule>
<acl:Rule>
        <acl:Type>Permit</acl:Type>
        <acl:Name>Rule01</acl:Name>
        <acl:Right>SELECT</acl:Right>
        <acl:On rdf:nodeID="v_foafview"/>
        <acl:When>Requester(Alice)</acl:When>
</acl:Rule>

ACL
<acl:Rule>
        <acl:Type>Permit</acl:Type>
        <acl:Name>Rule02</acl:Name>
        <acl:Right>ASK</acl:Right>
        <acl:On rdf:nodeID="v_foafview"/>
        <acl:When>TIME(clock,t) and (t&gt;8) and (t&lt;20)</acl:When>
</acl:Rule>
<acl:Rule>
        <acl:Type>Permit</acl:Type>
        <acl:Name>Rule03</acl:Name>
        <acl:Right>SELECT</acl:Right>
        <acl:On rdf:nodeID="v_foafview"/>
        <acl:When>IP(s,i) and NetUniv(i)</acl:When>
</acl:Rule>
<acl:Rule>
        <acl:Type>Permit</acl:Type>
        <acl:Name>Rule04</acl:Name>
        <acl:Right>CONSTRUCT</acl:Right>
        <acl:On rdf:nodeID="v_foafview"/>
        <acl:When>PlayRole=Friend</acl:When>
</acl:Rule>
</rdf:RDF>
```

Figure 6. Access Control List of foafview.txt

The Policy Enforcement Point of our proxy server applies the following recursive `enforce` function which takes as input a query. It returns an RDF graph or an SRD document depending on the query type.

```
Function enforce(q) : RDF graph or SRD doc
for v in s(q)
     If v is a view then
          v ← enforce(q(v))
return result of query q on s(q)
```

Performances

For evaluating the performances of our proxy, we used a RDF dataset containing over 400.000 statements about French cities. We showed that both the execution time of a user query (PEP) and the execution time of a security decision (PDP) are linear with the number of views which are not in the cache. We also showed that the execution time of a security decision is negligible compared to the time required for evaluating views.

6. Conclusion

In this paper, we have defined a new access control model for RDF data. Our main objective was to design a view-based model which was as convenient as the SQL security model. We believe we have obtained a powerful model allowing us to define flexible security policies consisting of dynamic security rules. We have successfully implemented our model in the framework of a proxy server for publishing RDF data. Our further works will follow the evolution of SPARQL. In particular, the SPARQL update 1.1 working draft [8] defines an update language for RDF graphs. Operations are provided to change existing RDF graphs. We are planning to include privileges corresponding to these operations in our security model. Another aspect, we could investigate is how to extend our model to support trust negotiation (see [7] for example). Indeed, in our model, access controls are very often made on the basis of subject attributes. These attributes can be digital credentials that are themselves sensitive objects which should be disclosed only after trust has been established between the requesting party and the service holding the resource. Trust can be gradually established between the two parties, through the iterative exchange of digital credentials. This means that we need security rules to define the security policy protecting not only the online resource (RDF data) but also the credentials exchanged during the negotiation.

Bibliography

[1] Yuan, E., Tong, J.: Attribute Based Access Control (ABAC) for Web Services. In Proc. of the IEEE International Conference on Web Services (ICWS'05).

[2] Reddivari, Pavan and Finin, Tim and Joshi, Anupam. Policy based Access Control for a RDF Store. In Proc.of the Policy Management for the Web Workshop series May 2005.

[3] Jain Amit and Farkas Csilla. Secure resource description framework: an access control model. SACMAT '06: Proc. of the eleventh ACM symposium on Access control models and technologies. Year 2006. USA.

[4] Fabian Abel , Juri Luca De Coi , Nicola Henze , Arne Wolf Koesling , Daniel Krause and Daniel Olmedilla. Enabling Advanced and Context-Dependent Access Control in RDF Stores. LNCS volume 4825/2008. October 2007.

[5] Jaehoon Kim, Kangsoo Jung and Seog Park. An Introduction to Authorization Conflict Problem in RDF Access Control. LNCS Volume 5178/2009. Knowledge-Based Intelligent Information and Engineering Systems. 2008.

[7] T. Yu, M. Winslett, and K. E. Seamons, Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies for Automated Trust Negotiation. ACM TISSEC, volume 6, number 1, February 2003.

[8] Simon Schenk and Paul Gearon. SPARQL UPDATE 1.1. http://www.w3.org/TR/sparql11-update/.

[9] Samarati Pierangela and Vimercati Sabrina De Capitani di. Access Control: Policies, Models, and Mechanisms. In Proc. of FOSAD '00. Year 2001.

[10] A. Seaborne. Rdql - a query language for rdf. tech. Rep., 2004.

[11] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, F. V. Vouton, and M. Scholl.Rql: A declarative query language for rdf. pp. 592–603, ACM Press, 2002.

[12] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation. 2008. http://www.w3.org/TR/rdf-sparql-query/.

[13] Apache tomcat. http://tomcat.apache.org/.

[14] Sesame, openrdf. http://www.openrdf.org/.

[15] RDF. Suite of W3C REcommendations. Home Page: http://w3.org/standards/techs/rdf#w3c_all.

[16] Damiani E., Vimercati SDC, Paraboshi S, Samarati, P. A fine grained access control for XML documents. ACM transactions on Information and System Security 5(2), 2002.

[17] Patrick Hayes and Brian McBride. RDF semantics. W3C Recommendation. http://www.w3.org/TR/rdf-mt/. 2004.

[18] Frédéric Cuppens and Alexandre Miège. Modelling Contexts in the Or-BAC Model. In 19th Annual Computer Security Applications Conference (ACSAC '03). 2003.

[19] FOAF Project. http://www.foaf-project.org/.

[20] D. Beckett and J. Broekstra. Sparql query results xml format. tech. Rep., 2008.

[21] E. Bertino, B. Catania, E. Ferrari and P. Perlasca. A Logical framework for Reasoning About Access Control Models. ACM transactions on Information and System Security, 6(1), February 2003.

[22] S. Jajodia, S. Samarati, ML Sapino ad VS Subrahmanian. Flexible Support for Multiple Access Control Policies. ACM Transactions on Database Systems, 26(2), June 2001.

[23] Fredéric Cuppens, Nora Cuppens-Boulahia et Meriam Ben Ghorbel. High-level conflict management strategies in advanced access control models. Workshop on Information and Computer Security (ICS'06). Timisoara, Roumania 2006.

[24] Oracle Database Management System. http://www.oracle.com