Additional mechanisms for rewriting on-the-fly SPARQL queries proxy

Arthur Vaïsse-Lesteven[1,2], Bruno Grilhères[2]

[1] GREYC, Team Modèle-Agent-Décision, University of Caen Basse-Normandie, France
arthur.vaisse-lesteven@unicaen.fr

[2] Team Information Processing Control and Cognition, AIRBUS GROUP, Val-de-Reuil, France
bruno.grilheres@cassidian.com

Résumé : A new approach for access management applied to semantic knowledge bases was explored by the WIMMICS team of INRIA. They built a proxy that according to an RDF security policy rewrites on-the-fly SPARQL queries. This work presents some extensions and upgrades applied to the original mechanism that corrects flaws in the application of the security policy.

Mots-clés : SPARQL, RDF, proxy, access management, security

1 Introduction

Denying or allowing access to a set of resources is a common problem in many computing fields. In the domain of the web of data, the open nature of the domain resulted in many systems that relied on trust in the user. This accessibility to data gave the impression to many institutions that data aren't safe in semantic repositories. In order to tackle this idea, in (Costabello *et al.*, 2012) the authors present a proxy that rewrites on-the-fly user queries according to security policies. However, the proxy presented in (Costabello *et al.*, 2012), despite a very good concept, presents weaknesses and reduces the expressivity of SPARQL, the standard language for requesting semantic data. This work presents the identified flaws and proposals to solve these problems.

2 State of the art

Several methods exist to secure a knowledge base exposed as a SPARQL endpoint on the web. The most used one consists of allowing total read-only access, but blocks all the update queries received. A more flexible approach is the use of the standard XACML technologies, which can be adapted for ad-hoc solutions. Deriving from these works, some recommendations now exist for adapting XACML to work with semantic technologies (Tyson, 2010). This approach, which consists of adapting previous technologies to new needs, was followed by many attempts to create new specific semantic data-access management technologies. Many of these systems, such as WAC (W3C, 2014) or AMO (Buffa & Faron-Zucker, 2012) or OAuth system (Tomaszuk & Rybinski, 2011), use an ontology to describe users, resources and user privileges, and so allow the representation of security policy using RDF. However, with these frameworks one can determine if a given access must be authorised or prohibited, but using only semantic technologies, can't filter access to only a part of the protected content dynamically. This is the functionality that the framework SHI3LD proposes.

3 The SHI3LD proxy

The SHI3LD proxy created by the WIMMICS team of INRIA protects data stored in a graph store according to a set of RDF security policies. When a query is sent to the proxy, it evaluates the request context. It consists mainly of information about the requester such as his identity, his company, his location... Depending on this context and policies, the proxy rewrites the query, and so grants access to some part of the graph store. This mechanism is illustrated in Figure 1 .

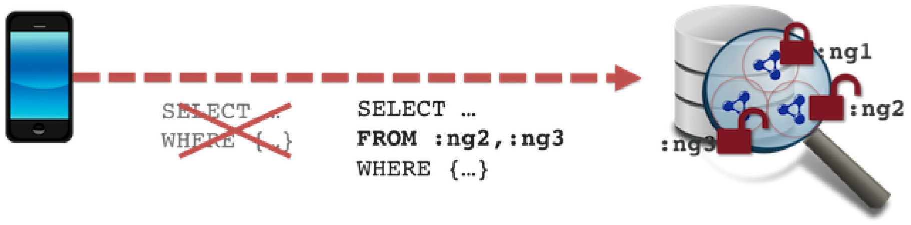


FIGURE 1 – General mechanism of SHI3LD (http ://wimmics.inria.fr/projects/shi3ld/)

In this example the requester tries to send a SPARQL request of type “SELECT”. The proxy uses all available information about the user’s context, and decides, in compliance with the policies, that among the three graphs stored in the knowledge base(“ :ng1”, “ng2” and “ :ng3”), the graph “ :ng1” isn’t available. So the proxy rewrites the user query by adding “FROM :ng2, :ng3” to it. In that way the user can only access authorized information and can’t even evaluate if information exists that he can’t access.
The authors defined their SHI3LD proxy as a three step process :

— Query Contextualisation : at request time, the user context of the query is associated to the user query to allow data protection. The user context is stored in the triple store.
— Access Policy Evaluation : using the user context, the proxy deduces a set of satisfied policies. Graphs protected by these policies are added to the set of the user’s accessible graphs. Given this set of authorised graphs, the query is rewritten.
— Query Execution : The rewritten query is sent to the triple store and the result returned to the client.

As policy satisfaction is context dependant, the same user addressing the same request to the proxy could obtain a different result if the context changes. The proxy dynamically adapts its comportment to the user. Additionally, the SHI3LD proxy instead calculates an authorised or prohibited access, it computes a result using the user privileges, and then returns it.
In the following sections this document presents some additional mechanisms proposed to extend the query rewriting process, and correct some flaws that concern the query rewriting process.

4 Flaws, limitations and proposals

4.1 Flaw caused by the additive-only nature of the process

During the SPARQL proxy query rewriting process all the accessible graphs are added to the user query in FROM clauses. An evident weakness comes from the fact that it is implicitly supposed that the users don't send any query that already contains a "FROM" clause that targets a graph of the triple store. In case of violation of this requirement, someone knowing a graph's URI can access any data in that graph, bypassing totally the proxy. This is illustrated in Figure 2 .

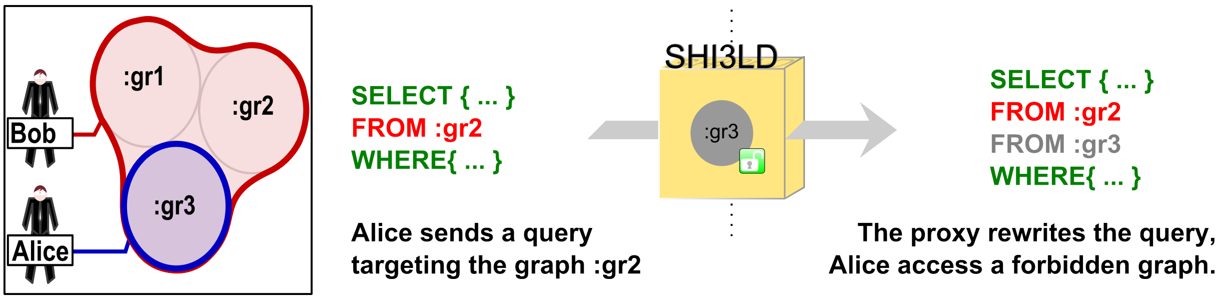


FIGURE 2 – Security flaw due to the additive-only nature of the rewriting process

In this example Alice sends a query that accesses a forbidden graph -on this example, as in all the following, the access is evaluated with a static context which entails that Alice can access to the blue graph (:gr3) and Bob to the red ones (:gr1, :gr2 and :gr3). The Access Policy Evaluation process using the policies concludes that Alice can only access the graph " :gr3", and so only graph " :gr3" is added to the query's range. So, the triple store protected by the proxy receives a query accessing data stored in the authorised graph and in the forbidden one. It results in Alice gathering forbidden data. An easy solution consists of explicitly blocking all queries specifying a set of target graphs. But, considering these facts :

— Targeting specific graphs is a part of the SPARQL 1.1 standard
— The RDF policies used with SHI3LD enforce structuring data using graphs
— (Costabello *et al.*, 2012) demonstrated that accessing fewer graphs improves the response time of queries

It seems logical to use the SPARQL language capability to target specific graphs in software, particularly when the location of the information in the triple store is known. In order to allow the use of such queries with the proxy while guaranteeing that access policies won't be violated, the rewriting process must remove targeted graphs whose access is forbidden, as illustrated in Figure 3

4.2 The zero target problem

There is a second easily identifiable problem that happens if the user can't access any graph at the end of the rewriting process.
This can occur in two cases :

— The user can't access any graph, and at the end of the process no target has been added.
— The user targeted only forbidden graphs, and the process ends after the removal of the last query's target.

Providing that the proxy uses the extension presented in the last part, the second use case obviously tends to appear more frequently than the other because users without any data accessible tend to be rare. This is show in Figure 3

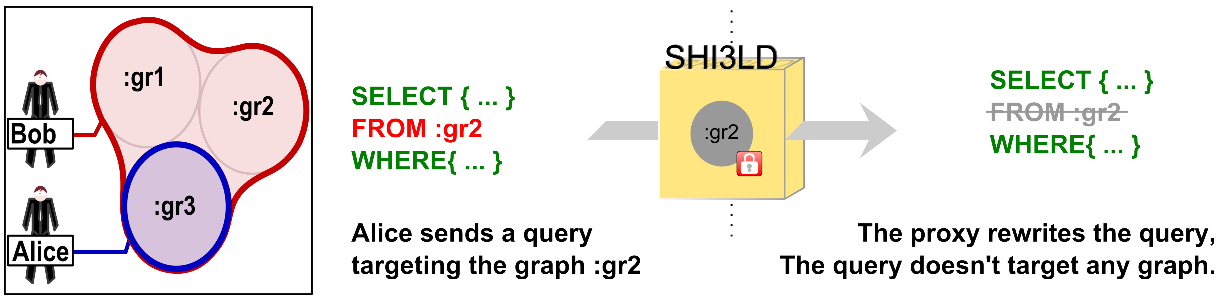

Bob
Alice
:gr1
:gr2
:gr3
SELECT { ... }
FROM :gr2
WHERE{ ... }
Alice sends a query targeting the graph :gr2
SHI3LD
:gr2
SELECT { ... }
~~FROM :gr2~~
WHERE{ ... }
The proxy rewrites the query,
The query doesn't target any graph.

FIGURE 3 – Security flaw due to the lack of description in the standard

Problems come with the fact that in that case the query transmitted to the triple store doesn't specify any target graph. In that case, the triple store executes the query against the default union graph, that is the graph that contains all the triples that aren't stored specifically in a graph. Because of the policies, all the triples are supposed to be stored in specific graphs in order to allow the expression of policies concerning these triples. Moreover, as they aren't in any graph, if such triples exist they aren't supposed to be accessible through the proxy as it provides access only to data stored in accessible graphs. Considering these facts, it is obvious that the result of a rewritten query that doesn't specify any target should be an empty set of results.
And here is the problem, the result computed by triple store on query reception depends on its implementation. The W3C standards do not define how a triple store is supposed to manage its default union graph. So, some implementations consider that the default union graph contains only triples not specifically stored in other graphs, as explained before. Meanwhile, others consider that the default union graph contains all the triples of the triple store with the graph dimension sliced out. These different behaviours have been observed on Fuseki and Sesame triple stores, both being standards and references for the domain in the category of open source triple store software. To tackle this problem of no-target queries, the solution proposed consists of adding a new graph $\mathbb{P}$.

Let $\mathbb{P}$ be the absorbing graph that always verifies :

$$\mathbb{P} = \varnothing$$

This property must be enforced by a policy that defines $\mathbb{P}$ as a read access only graph.

When, at the end of the accessible graph filtering, a query normally has to be executed on the union default graph of the triple store, the proxy instead forces its execution on $\mathbb{P}$ by adding it to the query range. As by construction, $\mathbb{P}$ is empty, the query result is necessarily an empty set of bindings.

4.3 Impossible to use both triple and quad patterns

Another problem was identified with the rewriting process of SELECT queries. The SPARQL SELECT queries are composed of two parts. The first one consists of a set of variables. This

set specifies the results returned by the query. Concerning the second part, it contains a pattern describing relations that have to be satisfied by any binding of variables to be correct and returned. Two type of pattern exist, triple patterns and quad patterns. Unlike a triple pattern that only describes a set of conditions to match bindings involving triples, a quad pattern describes relations about graphs that contain triples, and triples themselves. The SPARQL 1.1 standard describes how such patterns must be written and handled by triplestores.
The problem is that, as a side effect, the current mechanism invalidates the use of both triple and quad patterns on the triple store in a same query. It has been explained before that all the queries that pass through the proxy contain "FROM" clauses. The effect of a set of "FROM" clauses is the restriction of the range of the query to a graph corresponding to the union of all the graphs targeted by these clauses. That means that the graph dimension for the data accessed by the proxy has been removed. On other hand, when the query contains a quad pattern, "FROM NAMED" clauses are added. In that case the query is executed against a set of delimited and separated graph.

Queries using both triple and graph patterns in that case never have any correct binding, and so queries involving the two patterns never have any result. In order to allow the use of the full capability of the SPARQL language it's sufficient to add for each accessible graph, the "FROM" clause and the "FROM NAMED" clause targeting the same graph. The addition of these two clauses will result in a query executed on the union of the graph targeted and also on all the independent graphs targeted. This process is illustrated by Figure 4

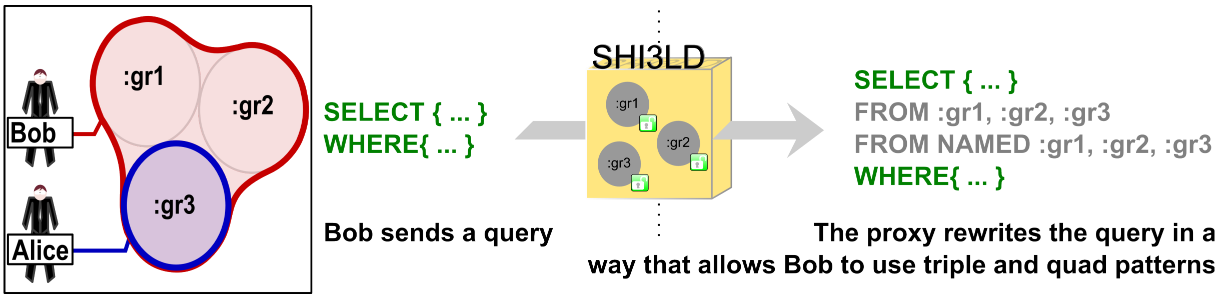


FIGURE 4 – Allowing quad patterns by adding FROM NAMED clauses

4.4 Federated queries problem

Finally, a last problem related to the SPARQL syntax comes from the standard capacity to handle federated queries. A federated query allows a query author to direct a portion of a query to a particular SPARQL endpoint. This may cause a flaw in the security of data. In fact when using a proxy like SHI3LD to protect a triple store, the protected triple store is configured to refuse all connections that don't come from the proxy. Indeed, the proxy can't rewrite the subquery associated with a distant endpoint as it doesn't protect this endpoint and so doesn't have to filter data that come from it. So, depending on your triple store software, using a federated query that requests the protected triple store using its local location can result in an unchecked access through the proxy to the data stored. This flaw is illustrated in Figure 5 .

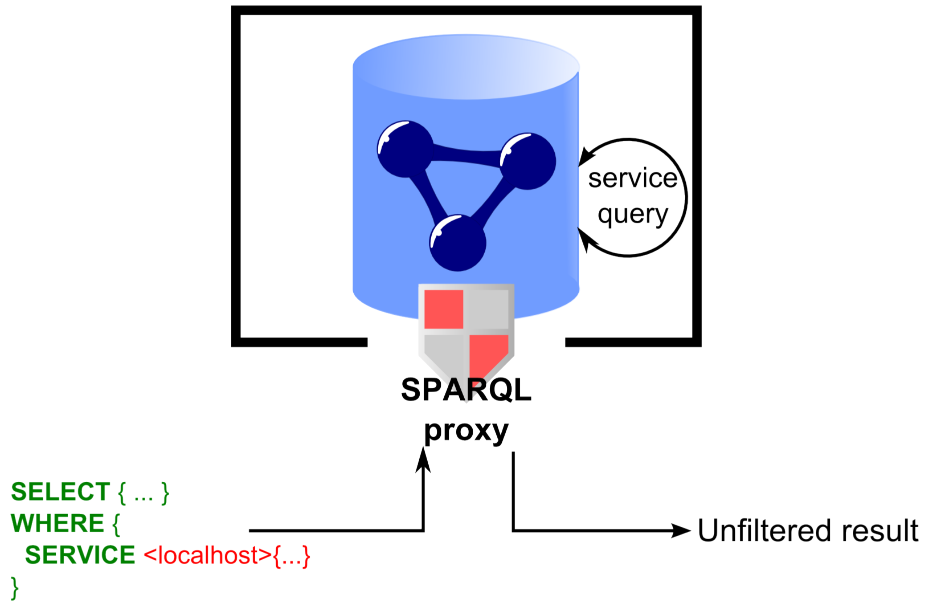


FIGURE 5 – Service used to bypass proxy

In order to correct this flaw the proxy must block queries that contain a SERVICE clause with the address of the protected triple store as external SPARQL endpoint. There is no reason to allow such use of the SERVICE clause ; patterns that concern the local triple store should be in the main query.
Finally, our works conclude that a improved algorithm for rewriting queries in a SHI3LD-like

proxy is :

```
Data : $Q$ : a SELECT SPARQL query, and $Ct$ : the context of the query
Result : $Q'$ : the query rewritten to match the policies
$Q' \leftarrow Q$
if $Q$ contains SERVICE clauses then
    for SERVICE clause $C$ of $Q$ do
        if Target($C$) $=$ protected triple store then
            return fail
$A \leftarrow \{$AuthorisedGraphs($Ct$)$\}$
$T \leftarrow \{$TargetedGraphs($Q$)$\}$
if $T = \varnothing$ then
    for graph $G$ of $A$ do
        AddFromFromNamedClause($Q', G$)
else
    for graph $G$ of $T$ do
        if $G \notin A$ then
            RemoveFromFromNamedClause($Q', G$)
$T \leftarrow \{$TargetedGraphs($Q'$)$\}$
if $T = \varnothing$ then
    AddFromPitGraph($Q'$)
return $Q'$
```

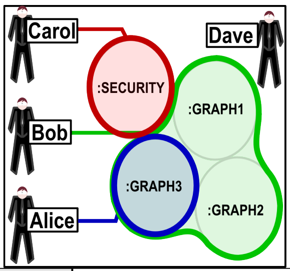
This processing allows the use of more capacity of the SPARQL language than the one presented by the WIMMICS team of INRIA while it still provides security for the data stored in the triple store.

5 Experimentations and Validation

During this work, given a set of users and a set of policies, we constructed a set of queries to cover SPARQL functionality. At execution time, these queries were executed, and then the obtained results compared to expected ones. The benchmark was executed each time on both Sesame and Jena triplestores. This approach revealed the previously identified flaws of the SHI3LD rewriting process, and conduced to the new process presented in this paper. Members of the team using only SPARQL requests weren't able to access forbidden data with this new process. A subset of these evaluation queries and the comparison between SHI3LD and our proxy is shown in Figure 6 .

6 Conclusion

Starting from an analysis of the core concept of SHI3LD, this document describes a problem coming from the additive-only nature of the query rewriting process. We propose adding a complementary subtractive approach that corrects this flaw. A second problem appears with



		With SHI3LD		With extensions	
User	**Base query**	**Rewritten query**	**Result**	**Rewritten query**	**Result**

Dave	**SELECT** { ?s ?p ?o }
	WHERE {
	?s ?p ?o;
	}
		SELECT { ?s ?p ?o }
		WHERE {
		?s ?p ?o;
		}
			{:GRAPH1, :GRAPH2, :GRAPH3, :SECURITY} | ∅
			It depends of triplestore
				SELECT { ?s ?p ?o }
				FROM :ABSORBING
				WHERE {
				?s ?p ?o;
				}
					∅

Carol	**SELECT** { ?s ?p ?o }
	FROM :GRAPH2;
	FROM :SECURITY
	WHERE{
	?s ?p ?o;
	}
		SELECT { ?s ?p ?o }
		FROM :GRAPH2, :SECURITY
		FROM :SECURITY
		WHERE{
		?s ?p ?o;
		}
			{:GRAPH2 :SECURITY}	**SELECT** { ?s ?p ?o }
				FROM :SECURITY
				WHERE{
				?s ?p ?o;
				}
					∅

Bob	**SELECT** { ?s ?p ?o1 ?o2 }
	WHERE{
	GRAPH ?g { ?s ?p ?o1}
	GRAPH ?g2 { ?s ?p ?o2}
	}
		SELECT { ?s ?p ?o1 ?o2 }
		FROM :GRAPH1,:GRAPH2, :GRAPH3;
		WHERE{
		GRAPH ?g { ?s ?p ?o1}
		GRAPH ?g2 { ?s ?p ?o2}
		}
			∅	**SELECT** { ?s ?p ?o1 ?o2 }
				FROM :GRAPH1,:GRAPH2, :GRAPH3;
				FROM NAMED :GRAPH1, :GRAPH2,:GRAPH3;
				WHERE{
				GRAPH ?g { ?s ?p ?o1}
				GRAPH ?g2 { ?s ?p ?o2}
				}
					QUAD PATTERN APPLIED ON THE 3 GRAPHS

Where {graph1,graph2} means : the union of graph1 and graph2

FIGURE 6 – Service used to bypass proxy

user queries without any graph target at the end of the rewriting process. This leads to, depending on the implementation of the triple store, a user being allowed to access all the data instead of getting a null result. The use of an absorbing graph added as a target for such queries has been recommended to solve this issue. Then, an analysis of the way triple stores work led us to identify a non-standard comportment with quad patterns due to the rewriting process. The addition of the FROM NAMED clauses during the addition of targeted graphs is sufficient to restore the normal comportment. Finally a security issue is identified coming from federated queries, that's why, if certain conditions are fulfilled, such queries must be blocked. This document ends with an algorithm proposal that contains all these improvements.

Références

BUFFA M. & FARON-ZUCKER C. (2012). Ontology-based access rights management. In *Advances in Knowledge Discovery and Management Studies in Computational Intelligence Volume 398 - 2012*.

COSTABELLO L., VILLATA S. & GANDON F. (2012). Context-aware access control for rdf graph stores. In *ECAI - 20th European Conference on Artificial Intelligence - 2012*.

TOMASZUK D. & RYBINSKI H. (2011). Oauth+uao : A distributed identification mechanism for triplestores. In *ICCCI (1)*.

TYSON P. (2010). Xacml and rdf. `https://wiki.oasis-open.org/xacml/XACMLandRDF`. Accessed : 2014-03-17.

W3C (2014). Web access control. `http://www.w3.org/TR/sparql11-query/`. Accessed : 2014-03-17.