Enforcing scalable authorization on SPARQL queries

Jörg Unbehauen
University of Leipzig
Augustusplatz 10
04109 Leipzig, Germany
unbehauen@informatik.uni-leipzig.de

Marvin Frommhold
eccenca GmbH
Hainstrasse 8
04109 Leipzig, Germany
marvin.frommhold@eccenca.com

Michael Martin
University of Leipzig
Augustusplatz 10
04109 Leipzig, Germany
unbehauen@informatik.uni-leipzig.de

ABSTRACT

With the adoption of the Linked Data Paradigm in the enterprise context effective measures for securing sensitive data are in higher demand than ever before. Exemplary, integrating enterprise systems containing millions of assets and fine granular access control rules with large public background knowledge graphs leads to both a high number of triples and a high number of access control axioms, which traditional methods struggle to process. Therefore, we introduce novel approaches for enforcing access control on SPARQL queries and evaluate their implementation using an extension of the Berlin SPARQL Benchmark.

1. INTRODUCTION

When deploying Linked Data technologies in enterprise contexts diverse information management systems are to be integrated, for example by extracting, transforming and loading their data into triple stores. Each source system may have a separate access control scheme put into place, which has then to be carried over to the RDF representation. For an example of low access control complexity consider data sets representing company background knowledge which originate from extracted thesauri or wikis and are typically available to large groups of users. For the complex case, consider a document management system holding 500k documents, each with individual access control rules. The extraction of meta data would yield about 15 million triples and directly assigning 100 users each to one half of the documents results in 25m document-to-user relations which, expressed as triples, would surpass the protected data in size. Typically, these access control rules are role based, which allows for better scaling with high user counts as access on protected resources is abstracted and granted via intermediate groups containing users with similar access privileges. Further, more restrictions may not only be direct user-to-document or user-to-group associations, but in fact depend on multiple attributes, such as authentication method (e.g. biometric or password), physical location or time-of-day.

In this paper we describe our graph-based access control approach that binds access control expressions to the context of RDF triples. Our hypothesis is, by storing a representation of the access conditions in the same store as the protected data we can rewrite SPARQL queries such that even in the face of complex authorization models both correctness and fast query execution is preserved. Additionally such an approach is vendor independent and can be used on standard SPARQL endpoints. To put our hypothesis to the test we implemented three different querying modes over the data, which are:

FROM The set of accessible graphs is conveyed in the *FROM* clause of a SPARQL query. Hence the set of accessible graphs is injected with each query into the protected end point. This allows us to counter-check our hypothesis, as here the access conditions are stored outside of the protected triple store. We also use this scenario for a comparison with the related approach shi3ld [5] as it uses the same mechanism.

QUAD-MAT The set of accessible graphs is computed once upon session initialization and stored inside the endpoint. All incoming queries are rewritten such that, additionally to the stated filters, URIs at the graph position of a quad must also exist in the previously stored set. QUAD-MAT therefore describes SPARQL quad based rewriting on a materialized set of accessible graphs.

QUAD-SUB Similarly to QUAD-MAT, URIs at the graph position are filtered. In contrast to QUAD-MAT, the set of accessible graphs is not precomputed, but determined using a subquery. QUAD-SUB therefore stands for SPARQL quad based rewriting using subqueries.

As a reference, we also discuss Virtuoso RDF Security[1] (**VOS-RDF-Sec**), Open Source version, as an example for a deep integration of security mechanisms into a triple store.

From a birds eye we can identify three important elements of the approach. First, the ontology which models the conditions a user and its session has to meet for accessing protected graphs (section 2.1). Second, a SPARQL query which yields the set of accessible graphs for a user, as described in section 2.2. Third, a SPARQL rewriting mode, which utilizes both model and query to actually enforce authorization by rewriting incoming SPARQL queries.

2. MODEL AND QUERYING CONCEPTS

This section describes how we model authorizations and how we query the authorization model in the different query

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

[1] http://docs.openlinksw.com/virtuoso/rdfgraphsecurity.html

modes. We assume that all triples reside in named graphs and form *g-boxes*[2]. An additional named graph contains authorization metadata on the other g-boxes. This authorization metadata is modelled and queried using the concepts subsequently described in this chapter.

By enforcing authorization with query rewriting we expose a SPARQL endpoint. For this SPARQL endpoint the default graph is the union of all graphs accessible by the user, as defined the authorization metadata.

2.1 Authorization Modelling

The authorization ontology was designed with the goals of allowing multi-attribute access conditions and a compact layout for fast querying. We reviewed existing ontologies, as discussed in section 4, and modeled the user session along the ideas of the shi3ld [5] *AccessEvaluationContext*. However, determining which access conditions apply to a user session fundamentally differs from shi3ld, where the access conditions are serialized as SPARQL ASK queries in the ontology. Each access condition requires the execution of its SPARQL ASK query. This prevents query execution as a subquery and in scenarios with high counts of access conditions can pose a significant overhead at the start of a user session.

We therefore designed an ontology that allows us to validate access conditions in a single query, which we demonstrate later in section 2.2. The concepts relevant for the authorization are illustrated in Figure 1. The main concepts of our vocabulary are `:Session` to declare the attributes of a user session and `:AccessCondition` to define required attributes for access on named `:Graph`s, where each `:Graph` identifies a g-box. A `:Group` is associated to a `:User` via the `:Session` indirection to model that the `:Group`- `:User` relation is valid at the time of `:Session` creation. The subproperties of `:requiresProp` pointing to a certain `:AccessCondition` define conjunctively all properties a `:Session` has to fulfill in order to make the `:AccessCondition` apply. In a similar fashion, properties of a `:Session` are modeled as subproperties of `:hasProp`.

```
1 @prefix ex: <http://example.org/auth/data/> .
2 <> a :AccessCondition ;
3  :requiresAuthMethod <http://dbpedia.org/resource/
       OAuth2>;
4  :requiresGroup ex:GroupUsers ;
5  :readGraph ex:PersonsGraph .
6 <> a :Session ;
7  :openedBy ex:UserA ;
8  :hasAuthMethod <http://dbpedia.org/resource/OAuth2>;
9  :hasGroup ex:GroupUsers .
```

Listing 1: User authentication model example

The example shown in listing 1 models an access condition which grants read access to the named graph `ex:PersonsGraph` for all users which are (line 3) authenticated via OAuth2 and (line 4) member of the group `ex:GroupUsers`. The example in listing 1 also lists a `:Session` (line 6) of `ex:UserA` (line 7) that is authenticated by OAuth2 and the association with the group `ex:GroupUsers`.

An intuitive evaluation of this example suggests that `ex:UserA` is allowed to read the data of `ex:PersonsGraph`.

2.2 Accessible Graph Determination

The user authorization named graph contains all defined access conditions and user sessions. Consequently the authorization named graph will be queried to determine the sets of named graphs G on which the user of a session has read access. Listing 2 shows the SPARQL query which determines G. The query consists of two subqueries, one which counts the number of all access conditions and the number of attributes fulfilled by the user session (listing 2 line 2-9) and another which counts the number of required attributes for each condition (listing 2 line 11-14). The superproperties (`:hasProp` and `:requiresProp`) are key to this approach, as they allow the definition of arbitrary conditions a session has to fulfill. By counting the number of conditions defined as subproperties between the `?session` and the `:AccessCondition ?cond` and checking for equality with the number of defined conditions of the `:AccessCondition` (listing 2 line 15), authorizations on `?graphs` are determined.

```
1  SELECT distinct ?graph {{
2      SELECT ?cond (COUNT(?attr) as ?joinCount)
3      WHERE {
4        ?has rdfs:subPropertyOf :hasProp .
5        ?req rdfs:subPropertyOf :requiresProp .
6        ?session :openedBy ex:UserA .
7        ?session ?has ?joinatt .
8        ?cond ?req ?joinatt .}
9      GROUP BY ?cond
10   }{
11     SELECT ?cond (COUNT(?attr) as ?attrCount) {
12       ?req rdfs:subPropertyOf :requiresProp .
13       ?cond ?req ?attr . }
14     GROUP BY ?cond}
15   FILTER (?joinCount = ?attrCount)
16   ?cond :readGraph ?graph .}
```

Listing 2: Query to determine readable graphs

2.3 Querying Modes

The querying mode determines how queries and the data of the triple store are modified for enforcing authorization. In this chapter we informally and briefly introduce the SPARQL rewriting methods using the terminology of [1].

The access control enforcement consists of two subsequent steps: first, the initialization (init) and second the querying operation. As part of the init process the backend is preparing session information, such as the authentication method and the groups of the user from an enterprise directory service. This session information is translated into triples using the vocabulary presented in section 2.1. We now describe the authorization flow enforced by the backend component when an authenticated user sends a SPARQL query to a protected endpoint.

FROM In this mode the access condition definitions and the session information are merged into an in-memory model in the authorization component. In the init step this model is queried to determine the set of accessible graphs G. Each incoming SPARQL query is hence analysed for the set of URIs in FROM F and FROM NAMED FN operators in both query and query parameters. As we defined the default graph to be the union of all graphs a user is authorized to access we therefore set $F' = G$. In case the FROM clauses were used, we filter both the FROM and FROM NAMED clause, such that $F' = G \cap F$ and $FN' = G \cap FN$.

QUAD-MAT Upon session creation, the list of accessible graphs is determined as in the *FROM* mode. An RDF representation of this list is then written into an additional named graph of the endpoint which is to be protected. Af-

[2] https://www.w3.org/2011/rdf-wg/wiki/Containers_of_Triples

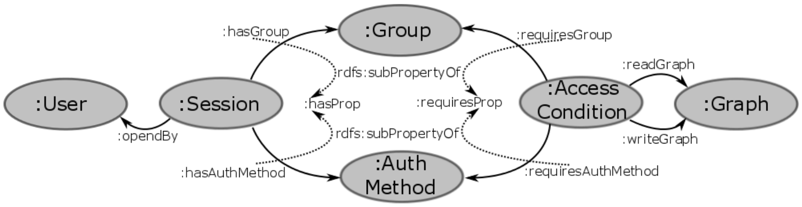

Figure 1: The authorization ontology with its major concepts.

terwards, for each triple pattern an EXISTS filter is added which ensures that the triple pattern is contained in an accessible graph by querying the materialized graph list as we show in listing 3.

```
... FILTER EXISTS {
GRAPH ?g { ?s a foaf:Person . } # original triple
     pattern
GRAPH <QuadMatGraph> {
<UserA> :canRead ?g . }
FILTER (?g IN ( LIST_OF_FROM_GRAPHS )) } ...
```

Listing 3: QUAD-MAT using EXISTS filter

QUAD-SUB During the initialization step, the session information of the user is written into the named graph containing the authorization metadata. Query rewriting is performed as shown in QUAD-MAT by adding an EXISTS filter for each triple pattern. The pattern consists of a quad pattern, a filter for the graph node of the quad and a subquery like in listing 2. Listing 4 illustrates an example of such an EXISTS filter.

```
... FILTER EXISTS {
GRAPH ?g { ?s a foaf:Person . } # original triple
     pattern
SELECT DISTINCT ?g {
# joins user session with ACs
# to determine accessible graphs
FILTER (?g IN ( LIST_OF_FROM_GRAPHS )) }} ...
```

Listing 4: QUAD-SUB EXISTS filter

VOS-RDF-Sec In this authorization scheme, the authorization component configures authorization facilities provided by the triple store. Like in mode QUAD-MAT, the list of accessible graphs is determined, translated into specific graph security statements and written to the Virtuoso instance. An incoming query is therefore conveyed unaltered to the SPARQL endpoint, the database connection is set to the requesting user to ensure correct query evaluation.

3. EVALUATION

For our evaluation we modified the Named Graph generation mechanism of the Berlin SPARQL Benchmark (BSBM) [3] to simulate three usage scenarios instead of one. BSBM simulates an e-commerce scenario, where *Products*, their *Reviews* and *Offers* and other related resources are browsed via a set of SPARQL queries. In the following we sketch our modifications to BSBM, the technical setup and summarize and interpret the results.

3.1 Benchmark and Scenario

Generating a benchmark data set is involves two steps. First, the protected dataset is generated and its data distributed among graphs according to the following scenarios:

BSBM Named Graph: In this scenario, which is part of the BSBM suite, all data is distributed in graphs with an average size of roughly 11,000 triples. The benchmark objects are associated with graphs based on a meta-properties of the data and consequently grouped together in moderately sized graphs.

Background Graph: In this scenario, a large graph together with high number of smaller graphs is generated. This scenario reflects scenarios where knowledge from restricted resources is used in conjunction with a public background knowledge graph. This scenarios is implemented by storing each Product resource in a separated separate graph and all the rest of the data in a background graph.

Resource Graph: Each benchmark resource is contained in a separate graph. Due to the BSBM data structure, the number of distinct subjects is equal to the number of graphs. We simulate with this distribution an extraction of a document management system, with each document having separate access conditions.

In a second step is the creation of users, groups and authorizations. For all scenarios and scale factors we generate 20 users in 10 groups and 10% of the users are admin users, which have read permission on all graphs.

The modifications to BSBM are available on github[3].

3.2 Implementation

We implemented all of the approaches with the help of the eccenca Linked Data Suite (eLDS)[4]. The suite consists of several components, of which the backend component can act as a proxy for SPARQL endpoints and forms the base for integrating all three querying modes. In case of the VOS-RDF-Sec mode we provide an implementation based on the RDF Graphs Security feature of the Virtuoso Open Source Server. For the other three modes, any SPARQL 1.1 compliant endpoint is supported.

3.3 Setup

We ran our benchmark on an Intel Xeon E3-1220 server with 8 GB of RAM, 128 GB SSD and all components (benchmark, auth component and triple store) running locally. Virtuoso 7.1 open source, was configured to use 3 GB of RAM and was used as the data store in all evaluations performed. As a reference point we further include a BSBM run without any authorization enforcement. In case of QUAD-MAT, QUAD-SUB and FROM, the benchmark was executed against the auth component. In case of VOS-RDF-Sec and the run without authorization the benchmark tested the Virtuoso server directly, bypassing the performance overhead imposed by the additional component.

[3] https://github.com/tomatophantastico/bsbm

[4] http://www.eccenca.com/de/produkte/linked-data-suite.html

3.4 Discussion

For the sake of conciseness we only present a summarization of the benchmark results in table 1.

QUAD-MAT and QUAD-SUB performance QUAD-MAT performs in the lower graph count scenarios considerably faster than QUAD-SUB by a ratio of 3.8 (table 1). In these scenarios the dynamic access control determination dominates the query cost. With higher graph counts this advantage diminishes and both techniques exhibit nearly the same performance. While slower than VOS-RDF-Sec, QUAD-MAT/SUB is able to execute all scenarios and is in all cases faster than FROM rewriting.

FROM Scaling the number of graphs and consequently of access conditions we observe that Virtuoso cannot process the high number of *FROM* clauses in the Background and Resource Graph scenario and does not yield results in these scenarios. Consequently, in table 1 these are marked as **n/a**. Exemplary, for a user who is able to view all graphs in the 10 million triples Resource Graph scenario we can extrapolate the query size to approx. 40 MB.

Virtuoso-RDF-Security performance VOS-RDF-Sec outperforms all other authorization approaches and only for the high number of graphs in the Resource Graph scenario with 10 million triples the gap between QUAD-MAT/SUB and VOS-RDF-Sec almost closes. Further, with a growing number of user-to-graph relations, loading times becomes longer. In the 10 million triples Resource Graph scenario the sequential load time was 201 minutes for the graph user relations, compared to 81 seconds for loading the actual data set. We want to stress here that this does not apply to the role-based access control of the commercial version of Virtuoso.

Scenario	*QUAD-MAT*	*QUAD-SUB*	*FROM*	*VOS-RDF-SEC*
BSBM Graph	2844	6401	10836	149
Background Graph	1764	6706	n/a	170
Resource Graph	2471	2844	n/a	1905
No Auth	n/a	n/a	n/a	104

Table 1: Total benchmark runtime in seconds for 500 runs on the 10 million triples data set.

4. RELATED WORK

A starting point for related work is the overview about core access control models and features for RDF in [6]. Categorized as enterprise access control schemes are *Mandatory Access Control (MAC)*, *Discretionary Access Control (DAC)* and *Role Based Access Control (RBAC)* which was initally introduced by [8]. Classifying our approach using this categorization model, we can highlight that we support *MAC*, *DAC* and *RBAC*. While mandated policies and user information are being received from a central authority (*MAC*, *DAC*) such as LDAP, the presented ontology (fig. 1) can be used to represent role concepts (*RBAC*) using the more abstract concepts *Session* in combination with the integrated superproperties.

An even more complex survey can be found in [7], which categorizes and analyses comprehensively the current landscape of well known approaches on enforcing access control on semantic data. A further approach is presented in [4], which can be sketched as a security architecture for enabling access control of ontologies. The authors re-use standard semantic web infrastructure with the goal of developing a security proxy that is able to rewrite SPARQL queries in order to observe defined access control policies. An approach likewise building on standard Semantic Web technologies is shi3ld [5], which was already discussed in section 2.1. With regards to the query rewriting technique, [2] proposes query rewriting in a similar fashion as we do, albeit using access control on triples and using SeRQL. Further worth noting is that both Openlink Virtuoso and Complexible Stardogfeature RBAC in their closed source products.

5. CONCLUSIONS

We introduced a minimalistic ontology which allows modelling of context and dynamic access conditions while exhibiting a compact querying footprint. Further, we describe two novel ways of rewriting queries, such that the access conditions are enforced, and created a freely available testbed for benchmarking graph based authorization schemes. Using this benchmark we demonstrate that our rewriting approach can effectively complement existing methods of query rewriting. We also demonstrate that the gap in performance compared to triple store integrated access control mechanism is, depending on the use case, tolerable. As both of our quad based query rewriting approaches only rely on SPARQL 1.1 we see them as an alternative, where stores with built-in access control methods cannot be deployed.

6. ACKNOWLEDGMENTS

This work was partly supported by grants from the German Federal Ministry of Education and Research (BMBF) for the LEDS Project under grant agreement No 03WKCG11C and the European Union's Horizon 2020 research and innovation programme for the SlideWiki Project under grant agreement No 688095.

7. REFERENCES

[1] SPARQL 1.1 Query Language. Technical report, W3C, 2013.

[2] F. Abel, J. L. De Coi, N. Henze, A. W. Koesling, D. Krause, and D. Olmedilla. Enabling advanced and context-dependent access control in rdf stores. In *ISWC*, 2007.

[3] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *IJSWIS*, 5, 2009.

[4] W. Chen and H. Stuckenschmidt. A Model-driven Approach to enable Access Control for Ontologies. *Wirtschaftsinformatik Proceedings*, 2009.

[5] L. Costabello, S. Villata, and F. Gandon. Context-Aware Access Control for RDF Graph Stores. In *ECAI*, 2012.

[6] S. Kirrane, N. Lopes, A. Mileo, and S. Decker. Protect Your RDF Data! In *JIST*. 2012.

[7] S. Kirrane, A. Mileo, and S. Decker. Access Control and the Resource Description Framework: A Survey. *Semantic Web Journal*, 2016 (to appear).

[8] R. S. Sandhu and P. Samarati. Access control: principle and practice. *IEEE Communications Magazine*, 32(9), 1994.