Poster: A Flexible Relationship-Based Access Control Policy Generator

Stanley Clark
Eindhoven University of Technology
Eindhoven, The Netherlands
s.clark@tue.nl

Nikolay Yakovets
Eindhoven University of Technology
Eindhoven, The Netherlands
hush@tue.nl

George H. L. Fletcher
Eindhoven University of Technology
Eindhoven, The Netherlands
g.h.l.fletcher@tue.nl

Nicola Zannone
Eindhoven University of Technology
Eindhoven, The Netherlands
n.zannone@tue.nl

ABSTRACT

A plethora of Relationship-Based Access Control (ReBAC) models have been proposed, varying in the types of policies they can express. This fragmentation has stifled the creation of a benchmark to directly compare the performance of ReBAC systems based on their common supported policies. To solve this problem, we propose RACON, a schema-driven, customisable ReBAC policy generator. RACON generates policies in an intermediate language subsuming the features required to encode existing ReBAC models. This language can subsequently be translated to popular ReBAC policy languages through an extensible translation module. Taking a view of ReBAC policies as graph queries, we implement translations into two popular graph query languages, namely Cypher and SPARQL.

CCS CONCEPTS

• **Security and privacy → Access control**.

KEYWORDS

Policy generation, Benchmark, Relationship-Based Access Control

ACM Reference Format:
Stanley Clark, Nikolay Yakovets, George H. L. Fletcher, and Nicola Zannone. 2022. Poster: A Flexible Relationship-Based Access Control Policy Generator. In *Proceedings of the 27th ACM Symposium on Access Control Models and Technologies (SACMAT) (SACMAT '22), June 8–10, 2022, New York, NY, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3532105.3535032

1 INTRODUCTION

Relationship-Based Access Control (ReBAC) [8] is an emerging access control paradigm supporting the specification of policies that naturally express relationships between entities in a graph. A variety of ReBAC models and languages have been proposed, many using custom ad-hoc algorithms [1, 5, 6] and others exploiting ReBAC's inherent graph-centric nature by relying directly on off-the-shelf graph databases [4, 10] to evaluate ReBAC policies specified as graph queries in languages such as Cypher [10] and SPARQL [4]. The policy languages used in these proposals are typically able to express different sets of policies based, for instance, on the graph topology, e.g. a policy granting a requester access to a forum if the requester knows a moderator of that forum.

To analyse and compare the performance of different ReBAC systems, one would require a policy benchmark encompassing the representative types of ReBAC policies and encoding such policies in the formats of the target systems. While similar policy benchmarks are available for other access control paradigms such as RBAC or ABAC via extensive policy generators, for instance extracting policies from real-world access logs [11], in ReBAC, each system is typically evaluated in the context of a set of hand-crafted policies suited to that system only. The absence of policy benchmarks for ReBAC is mainly due to the differences in expressiveness between ReBAC models and it thus remains difficult to directly compare the performance of ReBAC policy evaluation engines.

To address this problem, we propose a ReBAC policy generator, called RACON, which can generate large workloads of synthetic ReBAC policies based on several identified criteria from the research literature (e.g., [1, 5, 6]). To do so, we use an intermediate language based on queries over the property-graph model [3] to encode policies suitable to be translated to the vast array of ReBAC models. Our generation strategy is dictated by a user-defined configuration controlling the distributions of different types of policies.

2 RELATIONSHIP-BASED ACCESS CONTROL

A ReBAC model consists of *(i)* the set of information to be protected (often represented as a graph), or the *protection state* [6], *(ii)* the entities involved in the authorisation decision and *(iii)* a way to specify policies over the structure of the graph. The characteristics of these different components encoded by various ReBAC models [1, 4–6, 10] can be distilled into those presented in Table 1. Specifically, ReBAC models can be thought of as being able to specify a number of policy templates, which are a series of common patterns used in practice and exhibited by ReBAC policies. These templates may then be augmented to exhibit a number of additional features. For instance, a policy language may or may not support non-monotonic policies via negation or may require that both the requester and resource are used in the definition of access constraints, thus only being able to specify relational policies. The extracted characteristics can be used to model realistic ReBAC

SACMAT '22, June 8–10, 2022, New York, NY, USA

ACM ISBN 978-1-4503-9357-7/22/06.
https://doi.org/10.1145/3532105.3535032

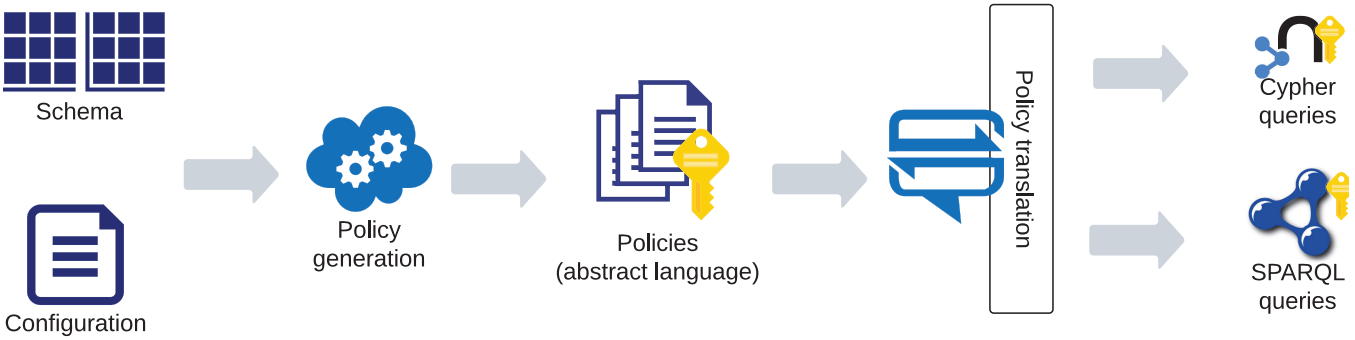

Figure 1: Overview of the RACON policy generation process

Table 1: ReBAC policy types and features

	Characteristic	Description
Templates	Common connectors [5]	Two users with k friends in common
	Clique [5]	A set of k users who all know each other
	Abstract path [1]	Two users connected by k steps
	All-of-type	A user knows all users of some type
Features	Non-finitary [6]	Transitive relations of arbitrary depth
	Non-monotonic [6]	Additional data reduces connectivity
	Owner-checkable [6]	Policy based on requester or resource
	Relational [6]	Policy based on requester and resource
	Restricted depth [5]	Users are friends via at most k people
	Any relationship	Predicate matching arbitrary edge label
	Property conditions	Filter on properties of nodes and edges

policies as they are based on characteristics extracted from real use-cases identified in the literature.

3 RACON POLICY GENERATOR

Using the policy templates and their features from Table 1, we have developed a ReBAC policy generator, RACON, able to create user-specified sized workloads of synthetic ReBAC policies. We have structured RACON around the architecture shown in Figure 1. We use a schema-driven approach to policy generation with an additional user-defined configuration representing the target distribution of policy features. This type of approach has previously successfully been used to construct robust and scalable benchmarks for database management systems [2]. We generate policies in an intermediate language and provide an extensible translation layer with two concrete implementations for both Cypher and SPARQL. RACON is available at https://gitlab.tue.nl/stanrogo/RACON.

Intermediate Language. Since the goal of RACON is to be able to generate ReBAC policies encoded in several target languages, we leverage an intermediate Datalog-style language in which policies are generated and from which they are translated to specific target languages. The intermediate language, designed to operate over property graphs [3], supports all the features of graph query languages shown in Table 2 and is therefore suited to encode all identified ReBAC characteristics. For instance, it supports Boolean output as opposed to typical graph query languages which are generally designed to retrieve sets of nodes.

As an example, the policy representing that a subject is allowed access to a resource when they have an (indirect) friend over the age of 20 can be expressed as:

$$\texttt{result}() \leftarrow \texttt{friendOfFriend}(\mathit{req}, x), \texttt{old}(x)$$
$$\texttt{friendOfFriend}(x, z) \leftarrow \texttt{friend}(x, y), \texttt{friend}^*(y, z)$$
$$\texttt{old}(x) \leftarrow \texttt{person}(x), x.\texttt{age} \geq 20$$

where variable *req* denotes the subject requesting access (i.e. the requester), $*$ represents the transitive closure of the `friend` relationship and x.age is a property of a vertex.

Policy Generation. The policy generation component generates policies in the intermediate language based on the four policy templates and features given in Table 1. The inputs are a *schema*, either manually specified or extracted from a database, specifying edge and vertex types together with their connections, and a *configuration* file specifying the number of policies to generate for each policy template, the target distribution each of the policy template features should follow and the node types of the requester and resource. When generating policies, we take into account the expected connections between the requester and resource vertexes as specified by the schema, only producing those policies for which there potentially is a match in some instantiation of the schema. The four policy types are then augmented with the properties from Table 1 according to the given Gaussian distribution in the format (*minimum*, *maximum*, *mean*, *s.d.*). Specifically, we augment the policy templates with: *(i)* properties of nodes and edges based on the schema; *(ii) any* relationship types which replace some specific edge types; *(iii)* regular expressions specifying paths of restricted or exact depth possibly containing transitive closures to replace edges of some specific type; and *(iv)* negations to generate non-monotonic policies. Additionally, we vary the presence of vertex variables representing the actors involved in a policy, i.e. the requester and resource, according to a user-defined distribution.

Policy Translation. The policy translation component translates policies from their representation in the intermediate language into a target policy language. Our system offers an interface using the

Table 2: Features of graph query languages to support ReBAC

Feature	Description
Property-graph	A property-supporting labelled graph
Homomorphic semantics	Allow repetition of edges or vertexes
Node inequality	To restrict homomorphic semantics
Regular path queries (RPQs)	Regular expressions over edge labels
Nested path negation	Express universal quantification
Parameterised queries	Specify runtime node bindings
Boolean output	Check existence of some witness
Correlated subqueries	Reference variables in negations
Any predicate	Match arbitrary edge labels
Non-path program	Queries on attributes only

visitor pattern to traverse individual elements of the intermediate language and produce an output string in the target language. Currently, our tool provides modules for SPARQL and Cypher.

Policies translated to Cypher adhere to the following structure:

```
MATCH (req), (res)
WHERE EXISTS {
    MATCH (a)-[:rpq]-(b)
    WHERE properties AND EXISTS {...} OR EXISTS {...}
} AND ID(req) = $req AND ID(res) = $res
WITH req LIMIT 1
MATCH (req) RETURN COUNT(req) = 1
```

Cypher queries are parameterised with the actors participating in the policy and instantiated at runtime in the context of an access request. Predicates and RPQs are encoded using individual `MATCH` keywords, while properties and (in)equalities defined in the policy are encoded as part of the `WHERE` clause. Each `EXISTS` subquery represents a grouping of policy predicates and their properties which can be combined using the `AND` and `OR` keywords to represent their conjunction and disjunction respectively.

Policies translated into SPARQL have the following structure:

```
ASK {
    ?req <hasType> reqType . ?res <hasType> resType
    FILTER EXISTS {
        { property . ?v1 rpq ?v2 . ...}
        UNION
        { property . ?v1 rpq ?v2 . ...}
    }
}
```

Unlike Cypher, SPARQL does not define a standard way to specify parameters. Therefore, it is up to the policy evaluation engine to replace instances of `?req` and `?res` with concrete values before executing the policy. The rest of the translation is similar to Cypher, with key differences being that the `OR` keyword is replaced by `UNION` and the types of the `req` and `res` variables are explicitly specified.

It is important to note that not every ReBAC policy language nor every graph query language supports all the features of the intermediate language. Specifically, not all ReBAC policies generated by our tool are supported in both SPARQL and Cypher. For example, SPARQL does not support matching a path regardless of the predicate label, the so-called 'any predicate' and does not support parameterised queries, requiring custom implementations to handle this and thus reducing portability. On the other hand, Cypher does not support the full range of RPQs required to be able to encode all ReBAC policies with certain features such as all non-finitary policies. Cypher can also merely simulate Boolean queries and requires a separate `MATCH` clause for every ReBAC policy predicate to simulate homomorphic semantics. When a given target ReBAC policy language does not support a particular generated policy, the policy translation component outputs an empty file.

4 CONCLUSION

In this work, we presented a ReBAC policy generator to generate user-defined synthetic workloads of ReBAC policies. RACON is designed to generate ReBAC policies incorporating all features of current ReBAC proposals. By using an intermediate language representation which can encode all ReBAC features, we are able to provide an easily extensible translation interface. This interface can be built upon to translate ReBAC policies into other languages such as EHL [9] or the upcoming GQL standard [7].

A typical use case of our tool is to generate a good mix of policies supported by a set of target languages for benchmarking purposes. To achieve such a mix, the user can control the parameters defining the distributions of the various ReBAC policy features. We have included a predefined configuration which we have found works well when using Cypher and SPARQL as the target languages.

Our work has a multitude of practical applications and benefits. First, it can be seen as a starting point from which to drive the inclusion of missing features into modern graph query languages. Moreover, it can be practically applied to assess the performance of evaluating ReBAC policies in popular graph databases that support Cypher, e.g. Neo4J, and SPARQL, e.g. Virtuoso.

REFERENCES

[1] Evangelos Aktoudianakis, Jason Crampton, Steve A. Schneider, Helen Treharne, and Adrian Waller. 2013. Policy templates for relationship-based access control. In *Annual International Conference on Privacy, Security and Trust*. IEEE, 221–228.

[2] G. Bagan, A. Bonifati, R. Ciucanu, G. H. L. Fletcher, A. Lemay, and N. Advokaat. 2017. gMark: Schema-Driven Generation of Graphs and Queries. *IEEE Transactions on Knowledge and Data Engineering* 29, 4 (2017), 856–869.

[3] Angela Bonifati, George H. L. Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. *Querying Graphs*. Morgan & Claypool Publishers.

[4] Barbara Carminati and Elena Ferrari. 2011. Collaborative access control in on-line social networks. In *International Conference on Collaborative Computing*. IEEE, 231–240.

[5] Yuan Cheng, Jaehong Park, and Ravi S. Sandhu. 2016. An Access Control Model for Online Social Networks Using User-to-User Relationships. *IEEE Trans. Dependable Secur. Comput.* 13, 4 (2016), 424–436.

[6] Philip W. L. Fong and Ida Sri Rejeki Siahaan. 2011. Relationship-based access control policies and their policy languages. In *SACMAT*. ACM, 51–60.

[7] Inc. JCC Consulting. 2022. *Graph Query Language GQL*. Retrieved April 13, 2022 from https://www.gqlstandards.org/

[8] Jorge Lobo. 2019. Relationship-based access control: More than a social network access control model. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9, 2 (2019), e1282.

[9] Edelmira Pasarella and Jorge Lobo. 2017. A Datalog Framework for Modeling Relationship-based Access Control Policies. In *SACMAT*. ACM, 91–102.

[10] Syed Zain R. Rizvi and Philip W. L. Fong. 2020. Efficient Authorization of Graph-database Queries in an Attribute-supporting ReBAC Model. *ACM Trans. Priv. Secur.* 23, 4 (2020), 18:1–18:33.

[11] Zhongyuan Xu and Scott D Stoller. 2014. Mining attribute-based access control policies from logs. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 276–291.