A Fine-grained Access Control Model for Knowledge Graphs

Marco Valzelli[a], Andrea Maurino[b] and Matteo Palmonari[c]
Department of Computer Science, Systems and Communication (DISCo), University of Milano-Bicocca, Viale Sarca 336, Milan, Italy

Keywords: Access Control, Property Graph, Knowledge Graph, NoSQL, Security.

Abstract: Nowadays Knowledge Graphs are a common way to integrate and manage all the information that an organization owns. This involve also sensitive domains like security, so the management of access control on these graphs became crucial. Due to their dimension, Knowledge graphs are often stored using NoSQL solutions, that have very poor support to access control. In this paper a distributed and secure Knowledge graph management system is presented. The system supports both open and closed access control and its architecture guarantees the management of very large knowledge graph.

1 INTRODUCTION

In 2012 Google introduced the Knowledge Graph (KG in the follow) as a new technology to improve its famous search engine. Basically, it is a graph that contains all the main named-entity of the world and the relations between them, so that the search engine is able to identify the subject of the query and suggest all object or features related to it. Since their introduction, KGs have been largely used by a lot of big and small companies (e.g. Amazon, Facebook and so on), but they are also used in several research domains as a way to describe in a common and shared way the knowledge of a given field. Several companies are now aggregating the knowledge stored in both structured and unstructured sources into a unique, comprehensive KG, for a better data management. It is worth noting that a KG can be considered as an evolution of the traditional data integration approach, and as a consequence the size of KG is bigger than previous solutions.

As the uses case of KG increase, they start to include also very sensitive data like for cybersecurity purposes (Piplai et al., 2019), in the military sector (Liao et al., 2019) the law enforcement sector (Szekely et al., 2015), but also civilian cases like the medical one (Rotmensch et al., 2017; Shi et al., 2017). For example in this last case, the KG includes all information related to doctors, patients, diseases and so on. Some people may access the KG in order to analyze correlation among clients but they cannot access to personal details of patients, while another person e.g. a doctor can access his patients' information but s/he cannot see information about others doctors. The explosion of KG raises new issues wrt. the preservation of relevant information from unauthorized access and their growing request for scalability at a higher level.

Currently, there is no common opinion about how to represent a KG. Among others, RDF is one of the most adopted description languages, SPARQL is the query language for RDF model and many systems for managing RDF data (also called triple store) are available. However there are proposals to implement an access control mechanism in RDF (Kagal et al., 2003; Abel et al., 2007; Kirrane, 2015), but they are not scalable nor flexible and, most important not implemented in commercial triple store. Recently, others graph models like the Labelled Property Graph model (LPG from now on) has been used to represent knowledge graphs.

In this short paper, we propose a solution for enabling access control of large KG in a scalable way. The scalability issue is solved by means of Janusgraph[1] a distributed and scalable property graph. We provide a systematic translation of RDF concepts in terms of property graph model. Then we define a access control method based on logical description of KG and users in terms of property graph model; our solution support a subset of SPARQL query lan-

[a] https://orcid.org/0000-0002-4028-3063
[b] https://orcid.org/0000-0001-9803-3668
[c] https://orcid.org/0000-0002-1801-5118

[1] janusgraph.org

Valzelli, M., Maurino, A. and Palmonari, M.
A Fine-grained Access Control Model for Knowledge Graphs.
DOI: 10.5220/0009833505950601
In *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications (ICETE 2020) - SECRYPT*, pages 595-601
ISBN: 978-989-758-446-6

guages by optimizing Gremlinator (Thakkar et al., 2018) a software layer able to translate a SPARQL query into a property graph query.

The paper is organized as follows: in section 2 we report the most important contribution of literature and industrial solution for the access control problem applied to graphs, while in section 3 we provide a more formal description of the problem. In section 4 presents the proposed access control method and in section 5 the implementation of the model is shown. Finally in section 6 we draw conclusion and discuss future works.

2 RELATED WORK

The great success of graphs as data model is very recent, but in the past we can find that trees, a particular type of graph, have been used as a data structure on which apply access control. This happened in the 2000 with XML, a format widely used to exchange information across the web. With the need to store data comes along the need to provide an access control framework (Damiani et al., 2002).

In this access control model users are identified with their IP address, an ID and a symbolic name. Those identifiers can be grouped in hierarchies and all this information is stored in XML too. Also users can be grouped, and for each group or user are assigned access rights. In turn, access rights give permissions on documents or document portions using XPATH specifications on the DTD. Finally, the rules are evaluated with defined precedence and propagation specifications between instance level and DTD level. Nowadays XML is also used as a serialization format for graphs(Brandes et al., 2013).

Starting from 2001 with the birth of the semantic web (Berners-Lee et al., 2001), the RDF semantic graph framework provided an efficient way to store big amounts of data in a graph format. RDF datasets usually are meant to be shared across the web, so no access control models have been provided. Since the users of RDF endpoints are often anonymous, the literature offers many proposals of context-based access control frameworks. One of these (Stojanov et al., 2017) also suggests an interesting mechanism for the creation of view on the graph based on the composition of allow/deny rules. The researches in this area also proposes to define policies through ontologies and to apply these policies using ontological reasoning. Research in this area also tried to address particular aspects of this technology like distribution (Kagal et al., 2003; Hollenbach et al., 2009), federation (Goncalves et al., 2019) inference of new information (Jain and Farkas, 2006) and possible solutions to the scalability problem (Abel et al., 2007). Also more traditional access control models like DAC have been implemented (Kirrane, 2015). Although you can find significant literature for access control in RDF Graphs (Kirrane et al., 2017), these approaches cannot be directly applied in other graph contexts as they focus on different aspects or heavily rely on ontological reasoning.

In recent years new storage models, called NoSQL, have come alongside the relational model. Unlike this one, that was made to strictly provide ACID properties, these models were developed to handle different use cases and huge amount of data following the BASE philosophy. However, these models were not designed with security and privacy criterion and still lack in this aspect (Sahafizadeh and Nematbakhsh, 2015; Alotaibi et al.,). We found that just a small part of these solutions support RBAC. Also notable is the case of Accumulo[2] where is provided access control at cells level.

The NoSQL wave also brought new interest on the graph model, due to its semi-structured nature that allows more flexibility and better performances in many use cases. The graph model supported by NoSQL products is very different from RDF. While it is based on triples, the property graph model is bases on nodes and edges on which you can specify particular attributes. We can find two kinds of property graph databases: the native ones, where data is physically stored in terms of nodes and edges, and the hybrid ones, that provides a property graph model but rely on a different storage model like the wide-column.

Neo4J is the most common native graph database and it implements the LPG model. It still has only basic security and access control features: it allows to define user role and their privileges[3], but not at a fine-grained level. Some initial step has been taken in this direction[4], but it still needs improvements to be able to cover the cases of open and closed policies.

The only significant work in the property graph context (Morgado et al., 2018) has designed an access control model defining how users or groups of users can access and manipulate data. However, it has some drawbacks like lack of flexibility and redundancy. In fact, you can only describe positive permissions and you have to define them for every node, so it's not clear how the model can be applied with inheritance and conflicts. Furthermore, this work and the neo4j

[2]https://accumulo.apache.org/

[3]neo4j.com/blog/role-based-access.-control-neo4j-enterprise/

[4]neo4j.com/blog/access-control-lists.-the-graph-database-way/

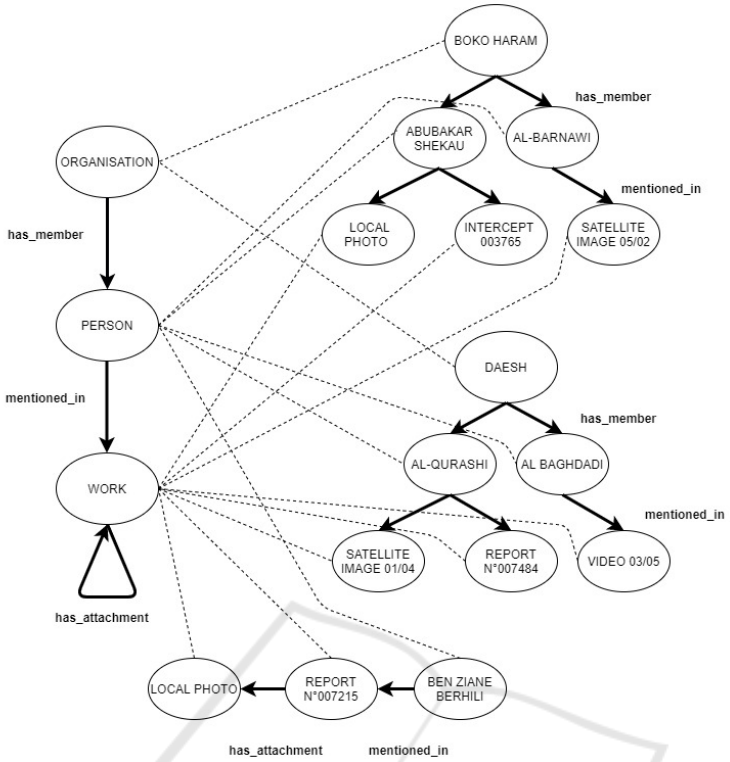

Figure 1: A practical use case of a source of general knowledge like DBpedia is extended with entities of the terrorism domain.

proposal has both the characteristic that they requires quite complex algorithm to be evaluated, so their scalability is in doubt. A different approach has been proposed in the Tinkerpop community[5] where the evaluation mechanism of the authorizations has been integrated in the GDMBS but it checks for access control metadata in every edge and node, so it needs a certain redundancy both in the stored data and in the interrogation, impacting on the performance of the system. In conclusion, the literature and the graph community proposed some suitable model, but it still has to be found a good trade-off between expressiveness, data overhead and evaluation time.

3 CONSIDERED USE CASES

In the past years the literature on access control, thanks to a great adoption in commercial DBMS, has defined a variety of policies one can implement.(Sandhu and Samarati, 1994). We focus on three main approaches for access control:

- **DAC:** Discretionary access control provides for a direct rights specification between users or groups of users and resources. This approach allows a great expressiveness, but at the same time lack in maintainability. For example, if you want to know every resource a specific user can access you would have to do a full scan query. In addition, you can adopt also open policy that allows specifying both denials and permissions. It is usually implemented with access control lists.
- **MAC:** Mandatory access control comes from the military context and provides access to the resources based on the clearance level of the user, following the need-to-know principle (closed policy): a subject should only be given access rights that are required to carry out the subject's duties.
- **RBAC:** Role-based access control is a more recent approach. It allows assigning rights to users based on their roles. When the user needs different rights, you only need to assign him to a new role without changing access properties to every resource he cannot access anymore.

These access control policies were designed to be mutually exclusive, but instead we want to combine and use them at the same time. Some effort has already been done in this direction (Jajodia et al., 2001). We extend it by adding more abstraction also over resources, borrowing the notion of classes from RDF

[5] archive.fosdem.org/2019/schedule/event/graph_access_control_tinkerpop

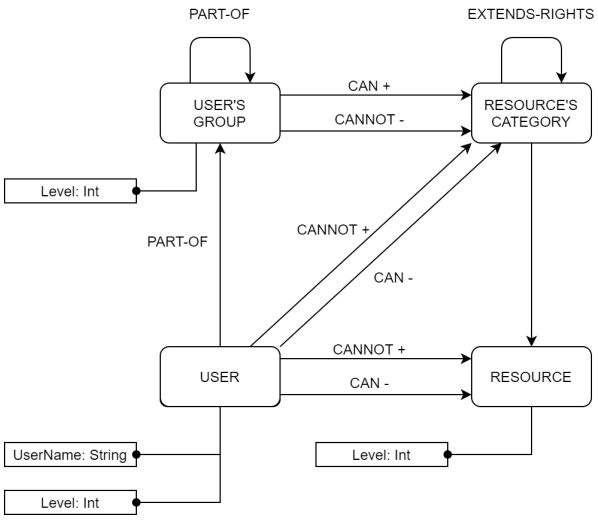

Figure 2: Graph pattern for the access control framework. ”Can” and ”Cannot” edges cannot be present simultaneously.

ontologies. In fact, groups of users and roles are ways to assign the same rights to more users at the same time, saving memory and maintainability time. We pursue the same objective over resources by giving the possibility to define access rights over entity classes and then propagate them using inheritance mechanisms.

For example, let's consider the case of a national security agency fighting terrorism, as described in image 1. For building the knowledge graph one has to start from a backbone of general knowledge. In this case, we use DBpedia [6], from which we use general classes like Person, Organization and Work. These classes can be used to classify and integrate sensitive information about terrorists and information about them. We call this extended graph the Domain Graph. It should be noted that we can use relations between classes as a way to extend access rights between instances. In fact, if a user has access to a criminal profile, probably she will have access also to his related documents (except for clearance). It is possible to represent as a graph also users, that can be grouped by departments of the agency or their roles. We call this part of the graph the User's graph. In this domain, we want to specify only what a user can view, with a need-to-know principle. That can be done on the base of both her role and groups, like with an RBAC policy. because probably different departments of the organization investigate different areas of the world, so they have to be able to view only a small portion of the graph. In addition, we may want to cover also particular cases (like temporary operation) that allows access to specific documents, like with a DAC. Plus, all those rules have to match a user's clearance level (MAC). With our work, we propose an access control model that cover all these cases.

[6]wiki.dbpedia.org/

4 ACCESS CONTROL MODEL

First of all, we have to point that the security of a computer system is composed of several ideal parts, that are an authentication service, an authorization database, an audit system, a security administrator and the access control model. In this paper we focus on this last part with the authorization database, assuming that the others are already present in the destination system. In access control systems a distinction is generally made between policies and mechanisms. Policies are high-level guidelines that determine how accesses are controlled and access decisions determined. Mechanisms are low-level software and hardware functions that can be configured to implement a policy. With our access control model, we pursue the goal to design a security mechanism that is reusable for many different policies, especially both the cases of open and closed policies. Our approach is focused on giving or not access to resource. More specific rights like own, delete, modify can be built on the top of our model.

The idea behind our model is to exploit all the flexibility of the LPG model so we do not give a rigid data structure, but only some graph patterns to follow, as can be seen in figure 2. We can identify three main parts in our model: the user's subgraph, the resource's subgraph and the authorization's edges between them.

In the user's subgraph users are represented by

nodes with properties containing username, password and clearance level. They can be grouped in groups by linking them to users' group nodes. This mean they will inherit the access rights of the group. In the same way users can be grouped by roles' node. Potentially these types of nodes can coexist, but this will lead to a more complex rights administration.

As we mentioned before, we use relation between classes as a way to extends rights between instances. In fact, it is a common situation that some objects are semantically dependents to another object, that we call resource category. Our intuition is to use this kind of relation between classes to automatically insert specific "extends_rights" edges between their instances.

Finally, we can specify access rights between users or user's groups and resource's classes with authorization edges. In our model we want to allow both open and closed policies, so these edges can have opposite meanings. With closed policies access' rights are usually assigned with the "need to know" principle, which is that a user has to be given access to the resources strictly necessary to carry out his duties, and everything else is forbidden. On the contrary, using open policies are specified resources where a user has not access and all that is not specified is allowed.

So the authorization edges must be used in different ways depending on the context: in the case of closed policies they must specify what an user can access, while in the case of open policies they specify what a user cannot access.

In addition our model provides for particular policy specification using exception edges, that are authorization edges between user and resource in contrast to those between users' groups and resources' categories. Imposing this constrain we ensure to allow expressiveness without leading to too much complexity.

So, after applying a closed policy, if you want to obtain all the nodes a user can access you need to navigate the graph up to the "part-of edges", search for "can" edges and then retrieve all the nodes belonging to that categories, then remove all the nodes linked by a "cannot" edge to the user.

In the same way to apply an open policy you know that a user can potentially access all nodes except those belonging to a category that is linked by a "cannot" edge to a group the user is part of. At the end you only have to re-add the nodes linked by exception edge "can" to the user. In both open and closed policies the nodes can also be filtered by the user's clearance level.

Determining which node a user can access with a traversal has many advantage:

- Graph model has great readability and also inheritance mechanism are easy to understand
- You can specify a great variety of policies with a few edges and attributes
- Full-scan are not needed as you can just use edges to reach resources you are or not allowed to access
- Maintainability is also simplified. For example, if a user wants to know which users have access to an object, she can exploit the reverse path used during the resource's access mentioned before

Now we try to clarify our model with a practical example, based on the previously presented Domain Graph of terrorism. It has been extended in 3 with the user's graph, where agents are grouped by nodes representing their roles and groups. The user's graph is linked to the Domain Graph with specific edges, and it is also integrated by AC edges itself. Let's clarify their meanings:

- Green edges represent the positive "can_view" permission.
- Red edges represent the negative "cannot_view" permission.
- Blue edges represent the extend rights relation.

This last relation can be manually specified between specific classes of the Domain Graph Ontology. For example, we specify that the relation "mentioned_in" extends the access rights from entity of type "Person" to entity of type "Work". Then we can use automatic procedure to add the access control attribute "extends_rights" to all the edges "mentioned_in" between entities of the classes defined before. From this specification we can deduce that:

- The department "Africa NSA" is allowed to view all Boko Haram organisation, but:
  - The object "Intercept 003765" cannot be seen from agent Paul and agent Patricia due to too low clearence level
  - The object "Satellite image 05/02" cannot be seen from agent Paul due to a "cannot" edge.
- The department "Middle East NSA" is allowed to view all Daesh organisation
- Agent Rick and agent Patricia are Director of their department, and this allows them to have full access to both organisation

In addition to this we can imagine that NSA organisation found a link between Daesh and the drug dealer Ben Ziane Berhily, who supplies daesh soldier with amphetamine. So agent Linda is allowed to view Ben Ziane data with an exception specified with the "can" edge.

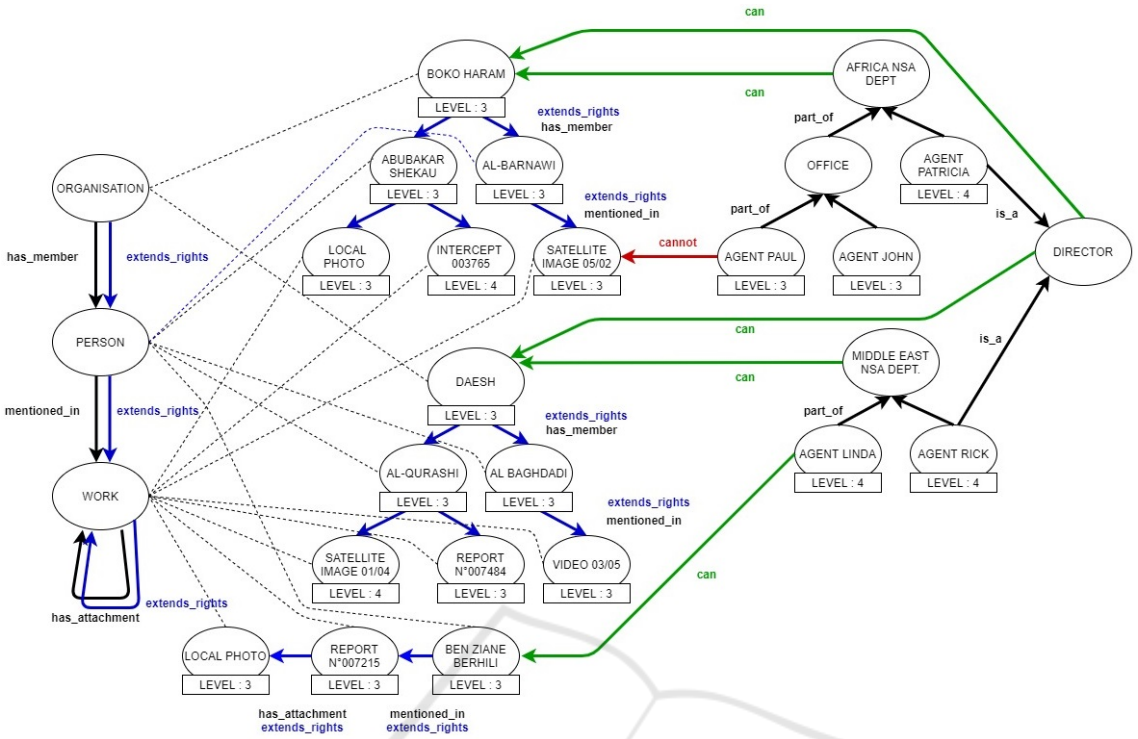

Figure 3: The before mentioned terrorism graph integrated with the users' Graph and the authorization edges.

5 IMPLEMENTATION

For the implementation of our access control model, we rely on Tinkerpop [7], which is an Apache project that provides several services and is compatible with fairly all the most common graph products. The most useful feature of this framework for our purposes is the SubGraphStrategy, which is a traversal strategy that allows to create a virtual subgraph based on given constraints. What this method really does is to provide an automatic mechanism that, given a user query, verify at every step if the initial constraints are satisfied.Since we use a pattern-based approach integrated with exceptions, we cannot use constraints based on attributes.

Our solution is to split the process into a two step. First, we retrieve all the IDs of the resource a user can access, then we use this list as a constraint for the traversal strategy. This prevents us to check for many different conditions at every step using a more immediate check like equality between IDs.

We now present a practical use example, based on the graph presented in section 3 and 4. If the access control strategy is not active, a general user can have full access to data. For example, if she asks for the names of criminals, the system will return all the nodes with class "Criminal" present in their "closure" attribute.

```
g.V().has('closure','criminal').values('name')
>
>[Abubakar Shekau, A-Barnawi, Al-Qurashi,
> Al-Baghdadi, Ben Ziane Berhili]
```

But if a specific user logs in to the system, agent Linda for example, our tool is able to build a personalized view. This is done in two steps. First, we make a query to identify the subset of nodes she has the rights to see, considering the specific user, her clearance level, and the before mentioned graph pattern (the whole query is omitted for space reasons):

```
allowedVertices = g.V().has('username','Linda')
.outE(). ... .has('level',P.lte(clearance))
.id().toList()
```

Then, a personalized view is generated giving as input the before extracted node ids:

```
sg=graph.traversal()
.withStrategies(SubgraphStrategy.build()
.vertices(__.hasId(P.within(allowed_vertices)))
.create())
```

Finally, the view is used to permit a transparent access to resources:

```
sg.V().has('closure','criminal').values('name')
>
>[ Al-Qurashi, Al-Baghdadi, Ben Ziane Berhili]
```

[7] tinkerpop.apache.org/

This way the access control uses the great amount of computational overhead at startup time, building the personalized views for all the users. However, these views has only to be generated once and then stored in-memory, leading to a very short response time when the user submits his query. At the same time, the user is not aware of the access control mechanism, making the process totally transparent. To prevent possible security issues, we have to protect script execution on Tinkerpop, or an expert user could easily bypass the SubGraphStrategy otherwise.

6 CONCLUSIONS AND FUTURE WORK

We find out that in the state of the art there are not efficient, flexible and general-purpose access control models for property graphs. We propose an approach based on graph traversal over specific patterns and on the creation of a subgraph using a Tinkerpop feature to address this issue. This is a preliminary model on the top of which it has to be build a more complete access control policy that includes write, delete and own rights. We left as a future work also an extensive test of the scalability of the model.

REFERENCES

Abel, F., De Coi, J. L., Henze, N., Koesling, A. W., Krause, D., and Olmedilla, D. (2007). Enabling advanced and context-dependent access control in rdf stores. In *The Semantic Web*, pages 1–14. Springer.

Alotaibi, A. A., Alotaibi, R. M., and Hamza, N. Access control models in nosql databases: An overview.

Berners-Lee, T., Hendler, J., Lassila, O., et al. (2001). The semantic web. *Scientific american*, 284(5):28–37.

Brandes, U., Eiglsperger, M., Lerner, J., and Pich, C. (2013). *Graph markup language (GraphML)*.

Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., and Samarati, P. (2002). A fine-grained access control system for xml documents. *ACM Transactions on Information and System Security (TISSEC)*, 5(2):169–202.

Goncalves, M., Vidal, M.-E., and Endris, K. M. (2019). Pure: A privacy aware rule-based framework over knowledge graphs. In *International Conference on Database and Expert Systems Applications*, pages 205–214. Springer.

Hollenbach, J., Presbrey, J., and Berners-Lee, T. (2009). Using rdf metadata to enable access control on the social semantic web. In *Proceedings of the Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK2009)*, volume 514, page 167.

Jain, A. and Farkas, C. (2006). Secure resource description framework: an access control model. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 121–129. ACM.

Jajodia, S., Samarati, P., Sapino, M., and Subrahmanian, V. S. (2001). Flexible support for multiple access control policies. *ACM Transactions on Database Systems (TODS)*, 26(2):214–260.

Kagal, L., Finin, T., and Joshi, A. (2003). A policy based approach to security for the semantic web. In *International semantic web conference*, pages 402–418. Springer.

Kirrane, S. (2015). Linked data with access control. In *Workshop on. pp*, volume 14, page 23.

Kirrane, S., Mileo, A., and Decker, S. (2017). Access control and the resource description framework: A survey. *Semantic Web*, 8(2):311–352.

Liao, F., Ma, L., and Yang, D. (2019). Research on construction method of knowledge graph of us military equipment based on bilstm model. In *2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS)*, pages 146–150. IEEE.

Morgado, C., Baioco, G. B., Basso, T., and Moraes, R. (2018). A security model for access control in graph-oriented databases. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 135–142. IEEE.

Piplai, A., Mittal, S., Joshi, A., Finin, T., Holt, J., Zak, R., et al. (2019). Creating cybersecurity knowledge graphs from malware after action reports.

Rotmensch, M., Halpern, Y., Tlimat, A., Horng, S., and Sontag, D. (2017). Learning a health knowledge graph from electronic medical records. *Scientific reports*, 7(1):1–11.

Sahafizadeh, E. and Nematbakhsh, M. A. (2015). A survey on security issues in big data and nosql. *Advances in Computer Science: an International Journal*, 4(4):68–72.

Sandhu, R. S. and Samarati, P. (1994). Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48.

Shi, L., Li, S., Yang, X., Qi, J., Pan, G., and Zhou, B. (2017). Semantic health knowledge graph: Semantic integration of heterogeneous medical knowledge and services. *BioMed research international*, 2017.

Stojanov, R., Gramatikov, S., Mishkovski, I., and Trajanov, D. (2017). Linked data authorization platform. *IEEE Access*, 6:1189–1213.

Szekely, P., Knoblock, C. A., Slepicka, J., Philpot, A., Singh, A., Yin, C., Kapoor, D., Natarajan, P., Marcu, D., Knight, K., et al. (2015). Building and using a knowledge graph to combat human trafficking. In *International Semantic Web Conference*, pages 205–221. Springer.

Thakkar, H., Punjani, D., Lehmann, J., and Auer, S. (2018). Two for one: Querying property graph databases using sparql via gremlinator. In *Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, pages 1–5.