Delegated attribute-based access control (DABAC) for contextual Linked Data

Riste Stojanov*, Sasho Gramatikov*, Milos Jovanovik*, Dimitar Trajanov*
* Faculty of Computer Science and Engineering, Skopje, Macedonia
{name.lastname}@finki.ukim.mk

***Abstract*—Security is an inevitable part of every information system. It is a cross-cutting concern that affects every part of the system. There is a constant trade-off between a secured system and convenient security management, and this management gets more demanding when the permissions are context dependent. The delegation of authorization is one way to make this process more convenient, by including and allowing multiple individuals to contribute.**

We provide a solution for combining multiple security rules such that data owners can delegate a part of their permissions over their data, and can validate the overall allowed data during the security permissions design process.

I. INTRODUCTION

The management of security in an information system is a lengthy and tedious process, where the security administrator should create access rules, represented as a security policies. The main role of the security manager is to define the permissions of the users in the system. However, this process becomes very complex when there is a hierarchy of numerous users with different access levels, since the manager has to be aware of all the peculiarities when defining the policies. In many situations the manager is in direct control of only the top-level users. In every system, there may be multiple security requirements relating different users and different contextual scenarios.

Another requirement that emerged with the wide penetration of the mobile devices and the Internet of Things is the contextual dependence of the security policies. Data has become more personal, more distributed, and hence, more vulnerable. Therefore, on one hand, our goal is to provide context-aware, attribute-based access control of the protected data, by using complex and diverse policies; on the other hand, we aim at simplifying the task of policy definition and reducing the human error factor in protecting the data.

We achieve the first goal by taking our previous work [1] as a starting point for context-aware attribute-based protection of the data by combining multiple policies with different priorities. In order to achieve our second goal, i.e. to simplify the access-control definition, we extend the LDA platform by proposing delegated access-control. The security manager defines policies only for the top-level security users and lets them delegate a subset of their access rights to the next level of subordinate users. The subordinate users can proceed with the same principle of delegation. Each user with delegated rights defines new policies that are combined with all inherited policies up to that level. With this, the complexity of securing the system is distributed to a larger number of users which imposes fewer errors, better control and protection of the data. Moreover, the approach guarantees that users cannot grant access to data that they themselves are not allowed to access. The possibility of potential human errors in defining the policies is further reduced by employing the validation tools of the LDA platform.

II. RELATED WORK

The authorization process generally enforces the policies, which are formalized requirements.

Policy enforcement can be grouped into three main categories:

- **Permitted data filtering/annotation** is storage level protection. It is most commonly implemented using a graph that contains the permitted data [4, 5] or by permitted data annotation [6]. This enforcement method introduces significant performance degradation if it is executed on every request [2, 4].

- **Query rewriting** enforcement adds additional constructs to each query that enters the system in order to ensure that only the permitted resources can be obtained. This approach requires complex algorithms for the rewriting process, and thus it needs extensive correctness and performance testing, as discussed in [2].

- **Per action** enforcement is the most commonly used, due to its simplicity. It permits or denies the domain actions. The downside of this enforcement method is that its policies are not able to filter out the forbidden resources.

The policy format definition is generalized in [2] as:

```
<Subject, Resource, Access Right>
```

The *Subject* from this tuple describes for whom this policy will be active. The *Resource* is intended to describe the resources that are protected by the policies, while the *AccessRight* describes whether the policy allows or denies for certain action.

The access rights should define whether the policy *permits* or *denies* access to an action to interact with resources on behalf of a requester in a given context. It is defined as a condition expressiveness in [3], with additional obligation value that requires something to be executed. This paper focuses only on access control, and the obligation is left out to be incorporated in the business logic.

The Access Control Model is the most widely used description for authorization systems. It can be closely related to the previous policy formalization. The following access control models can be find in the literature (including their combinations): - **MAC: Mandatory Access Control** [7] defines a policy for each combination

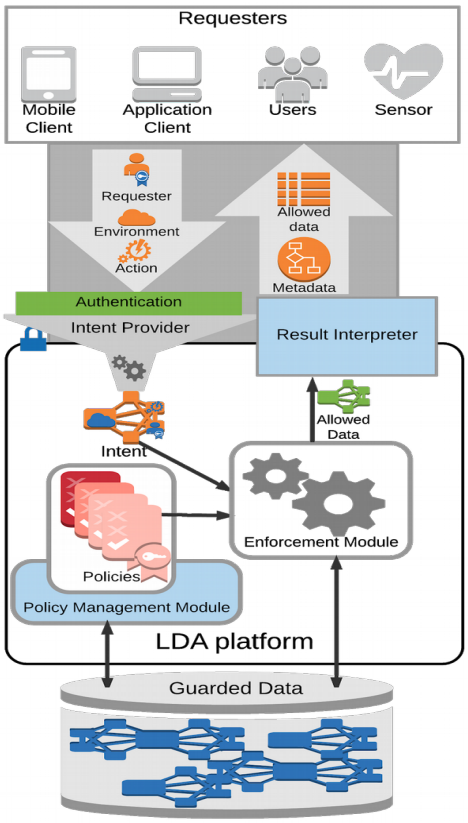

Figure 1. The LDA platform architecture

of subject and resource. The policies are stored and managed by central authorities. In the context of semantic web authorization, the approaches presented in [10, 11] can be categorized as MAC.

- **DAC: Discretionary Access Control** [8] enables policy delegation, and they are stored and managed in a distributed manner.

- **RBAC: Role Based Access Control** [9] introduces user grouping into roles [5, 2], which significantly simplifies the policy management, through the reduction of the number of policies.

- **ABAC: Attribute Based Access Control** [12] is another model that focuses on the requester specification and provides more flexibility through dynamic grouping of the requesters using their attributes [13, 14, 15]. This model is suitable in the scenarios where dynamic separation of duty is required.

- **VBAC: View Based Access Control** [16] model groups the resources into views, and authorizes the requesters based on these groups [5, 15, 4, 17]. Similar to RBAC, it simplifies the policy management through the reduction of the number of policies.

- **CBAC: Context Based Access Control** [18] model enables contextual policy definition [14, 15, 19, 6].

The correctness of semantic authorization systems is analyzed in [2], where the query rewriting implementation correctness is tested against a permitted data annotation approach. However, the possible errors in the policy design and definition processes are not considered in this paper.

When both permit and deny policies are available, their definition is simpler, but conflicts may arise [3, 17, 15]. There are various ways to solve these conflicts, such as: default behavior [19, 17], meta-policies [11], priorities [20], and detection and prevention [14, 4].

The main challenge in each system that needs to be secured is the trade-off between the protection and the maintenance convenience. The policy management process requires a flexibility to protect an arbitrary part of the data, for every particular user or group of users in a specific context. Furthermore, the policies are managed by human beings, and as such, are prone to errors and mistakes. Therefore, a secure policy management system should provide design-time security rules validation. All these issues are addressed in our previous work [1]. However, this work does not provide a convenient delegation of access rights. There are systems that allow access rights delegation [2, 3], but they are focused on delegating access based on user roles, which is not as flexible as using the attributes.

III. Linked Data Authorization (LDA) Platform

In this paper we are extending our previous work [1] by allowing policy combination such that the users can delegate rights for their data to other entities. The architecture of the platform is shown in Figure 1. The goal of the platform is to ensure the security requirements defined as policies against the *Guarded Data*. This platform accepts the user requests that hold the information about who the requester is, from which context it is accessing the platform and the action it is intending to perform. This request is first intercepted by the *Intent Provider* component, which authenticates the user based on the provided information in the request and builds a semantic representation of the request referred to as *Intent*.

The policy *Enforcement Module* provides the machinery that ensures the appropriate data protection. It first combines the configured security policies relevant for the current *Intent*, and then filters only the data allowed in that given scenario. The results are then handled to the *Result Interpreter* component, which serializes the results in the requested format.

The *Policy Management Module* is the administrative part of the system that allows the security officers to specify the policies. During the policy design time, this module provides evaluation of the data being protected by each policy and presents the relevant Intents for which the policy will be activated. It also enables detection of potential conflicts among the defined policies, as well as detection whether a portion of the data is not protected. Figure 3 shows the policy administration interface that also allows design-time validation of the allowed data for the policy for different Intents.

In this work we also defined a policy language that extends the SPARQL query language with constructs that describe whether a data portion is allowed or denied for a certain *Intent*. The term *Intent* is used to represent the condition against the requester and its context. An example policy that describes the ability of our policy language to provide a contextual protection with respect to the user attributes, is shown in Figure 2. This policy corresponds to the following security requirement:

```
ALLOW READ { ?s ?p ?o ?g }
WHERE {
  GRAPH <http://intent> {
    ?r a int:Requester.
    ?ag a int:Agent; int:address ?ip.
    ?ip int:network ?n
  }
  ?r sm:works at ?v8.
  ?v8 sm:network address ?n.
  ?v9 sm:has doctor ?r; sm:for patient ?v11.
  ?v10 sm:owner ?v11.
  GRAPH ?g {
    ?s sm:sensor ?v10; ?p ?o
  }
} PRIORITY 7
```

Figure 2. Example Policy

The doctors can read all sensor observations of their patients, but only from their hospital network.

The ability to model this requirement shows that this policy language is able to link the requester with its context and the data that is being protected in the same policy, which significantly improves the administration effort. The graph <http://intent> shown in this policy is the semantic representation of the user *Intent* injected as a temporal graph during the enforcement of the policies for each request.

IV. DELEGATING THE AUTHORIZATION

The main extension in this paper is in the *Policy Management Module*. Here, we allow each user to define a policy that delegates the access of its allowed data to other users, using the standard interface and policy syntax. However, in this scenario, some user may give access to data that is not allowed for him/her. In order to prevent this scenario, each policy is stored by removing the data that is not allowed for the given user.

We are going to explain the policy modification using an example, where the data owner has gained access to the data D_{own} obtained with the query q_{own} from the guarded data D_{all}. The query parts that are discussed here are the one that describe the protected data, i.e. the *WHERE* block from the policy without the *GRAPH <http://intent>* block element. Let this user create a policy that protects the data D_d with the query q_d for the intents that are suitable for the query i_d. In this scenario, the resulting policy should protect the data D_d that intersects with D_{own}. Since the SPARQL syntax does not support the intersection operation, we are using the following set transformation which can be implemented with SPARQL operators:

$$D_d \cap D_{own} \Leftrightarrow D_d \setminus (D_{all} \setminus D_{own})$$

The right-side expression can be implemented with the following SPARQL construction:

$$\begin{aligned} &\dots \textit{WHERE } \{ \\ &\quad q_d \ \textit{MINUS} \{ \\ &\qquad q_{all} \ \textit{MINUS} \{ \ q_{own} \ \} \\ &\quad \} \\ &\} \end{aligned}$$

where the q_{all} query selects all the data from the dataset using the pattern *?s ?p ?o*.

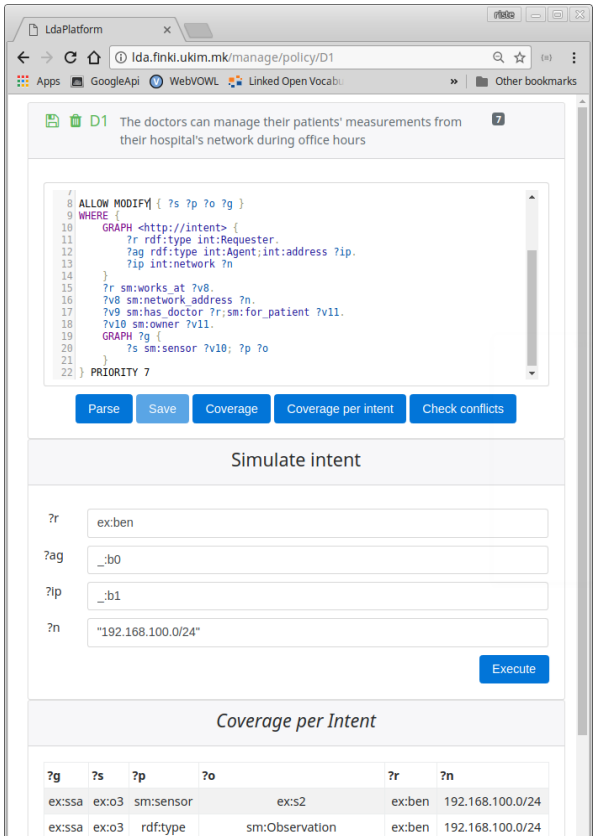

Figure 3. Example Policy

The *Policy Management Module* transforms each policy that delegates access such that it has the following WHERE block:

$$\begin{aligned} &\dots \textit{WHERE } \{ \\ &\quad i_d \ . \\ &\quad q_d \\ &\quad \textit{MINUS} \{ \\ &\qquad q_{all} \ \textit{MINUS} \{ \ q_{own} \ \} \\ &\quad \} \\ &\} \dots \end{aligned}$$

This transformation allows the *Enforcement Module* to operate without modification, and when a certain intent activates the policy (satisfies the query i_d), it will allow/deny the data $D_d \cap D_{own}$. In this case, the portion of the data D_{own} will be constructed with respect to the current intent, which will preserve the context awareness of the authentication process. If one of the policies that construct D_{own} is not suitable for the given intent, it will return an empty set of triples, and therefore it will not influence the final result. The *Enforcement Module* uses the i_d part of the delegated policy for optimization, where it filters only the applicable policies for the intent in advance, and then activates and combines only the ones that are suitable in the given context.

V. Discussion

The proposed extension to the existing LDA platform enables delegation of the configuration effort from the owners toward other users, which significantly distributes the maintenance effort.

The Enforcement module provides implicit security, since its implementation first creates a temporal dataset that contains the allowed data in the given context, and then executes the requests against this data. This implementation has a performance trade-off, since it is not suitable in a scenarios when there are huge amounts of data allowed for the user. However, in most of the real-life application, each regular user has access only to a portion of the overall data, where this approach has acceptable performance, as discussed in [1].

Another unique feature provided with this extension is the possibility for a user to inherit data from multiple owners, since each owner embeds its ownership rights in the delegating policy.

As in every long-leaving application, the policies may change over the time. Our extension keeps track of the policy delegation path, storing information about the user that has created it. Therefore, every time there is a modification of a policy, all policies originating from the creator are updated. In the worst-case scenario, when the system administrator updates a general policy, all delegated policies are updated. In order to prevent some major downtime, all requests that have started will use the old version of the policies, while the new requests will wait for the updated version of the policies. However, since the policy update is performed in-memory, this operation finishes in milliseconds, even if there are thousands of policies in the system.

VI. Conclusion

In this paper, we provide a flexible policy language that enables protection to arbitrary data parts in relation to the requester and its context. The design-time policy validation ensures the proper protection of the data. This paper focuses on the policy transformation in the process of delegation of the access rights. The intent query part of the policy provides that the delegation can be temporary for a given context with respect to the requester attributes, while the protected data part ensures that the data owner can only specify policies for a subset of its owned data.

The policy enforcement module provides activation and combination of the defined policies such that the data owners can have a convenience to protect multiple parts of their data with separate policies, that can be validated during the design time.

Acknowledgment

The work in this paper was partially financed by the Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University in Skopje, as part of the “DataGEM: Data science Based Global Economy Modeling and Forecasting” research project.

References

[1] R. Stojanov, S. Gramatikov, I. Mishkovski, D. Trajanov. "Linked Data Authorization Platform." IEEE Access 6 (2018): 1189-1213.

[2] S. Kirrane, A. Mileo, and S. Decker. "Access Control and the Resource Description Framework: A Survey." Semantic Web 8.2 (2017): 311-352.

[3] S. Kirrane, A. Abdelrahman, A. Mileo, S. Decker. "Secure Manipulation of Linked Data." International Semantic Web Conference. Springer, Berlin, Heidelberg, 2013.

[4] H. Muhleisen, M. Kost, J. C. Freytag. "SWRL-based access policies for linked data." Procs of SPOT 80 (2010).

[5] S. Dietzold, S. Auer. "Access control on RDF triple stores from a semantic wiki perspective." ESWC Workshop on Scripting for the Semantic Web. 2006.

[6] S. Franzoni, P. Mazzoleni, S. Valtolina, E. Bertino. "Towards a fine-grained access control model and mechanisms for semantic databases." IEEE International Conference on Web Services (ICWS 2007). IEEE, 2007.

[7] P. Samarati, S. C. de Vimercati. "Access control: Policies, models, and mechanisms." International School on Foundations of Security Analysis and Design. Springer, Berlin, Heidelberg, 2000.

[8] R. S. Sandhu, P. Samarati. "Access control: principle and practice." IEEE communications magazine 32.9 (1994): 40-48.

[9] R. S. Sandhu. "Role-based access control." Advances in computers. Vol. 46. Elsevier, 1998. 237-286.

[10] H. Hollenbach, J. Presbrey, T. Berners-Lee. "Using rdf metadata to enable access control on the social semantic web." Proceedings of the Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK2009). Vol. 514. 2009.

[11] L. Kagal, T. Finin, A. Joshi. "A policy language for a pervasive computing environment." Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks. IEEE, 2003.

[12] T. Priebe, W. Dobmeier, N. Kamprath. "Supporting attribute-based access control with ontologies." First International Conference on Availability, Reliability and Security (ARES'06). IEEE, 2006.

[13] O. Sacco, J. G. Breslin. "PPO & PPM 2.0: Extending the privacy preference framework to provide finer-grained access control for the web of data." Proceedings of the 8th International Conference on Semantic Systems. ACM, 2012.

[14] A. Toninelli, R. Montanari, L. Kagal, O. Lassila. "Proteus: A semantic context-aware adaptive policy model." Eighth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'07). IEEE, 2007.

[15] L. Costabello, S. Villata, O. R. Rocha, F. Gandon. "Access control for http operations on linked data." Extended Semantic Web Conference. Springer, Berlin, Heidelberg, 2013.

[16] P. P. Griffiths, B. W. Wade. "An authorization mechanism for a relational database system." ACM Transactions on Database Systems (TODS) 1.3 (1976): 242-255.

[17] G. Flouris, I. Fundulaki, M. Michou, G. Antoniou. "Controlling access to RDF graphs." Future Internet Symposium. Springer, Berlin, Heidelberg, 2010.

[18] A. Corrad, R. Montanari, D. Tibaldi. "Context-based access control management in ubiquitous environments." Third IEEE International Symposium on Network Computing and Applications, 2004.(NCA 2004). Proceedings.. IEEE, 2004.

[19] F. Abel, et al. "Enabling advanced and context-dependent access control in RDF stores." The Semantic Web. Springer, Berlin, Heidelberg, 2007. 1-14.

[20] V. Kolovski, J. Hendler, B. Parsia. "Analyzing web access control policies." Proceedings of the 16th international conference on World Wide Web. ACM, 2007.