Representing Attribute Based Access Control Policies in OWL

Nitin Kumar Sharma
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi, India-110016
Email: mcs142867@cse.iitd.ac.in

Anupam Joshi
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County, Baltimore, MD
Email: joshi@umbc.edu

***Abstract*—Attribute Based Access Control (ABAC) models are designed with the intention to overcome the shortcomings of classical access control models (DAC, MAC and RBAC) and unifying their advantages. In ABAC, the access control is provided based on generic attributes of entities. Many organizational security policies condition access decisions on attributes. OWL can be used to formally define and process security policies that can be captured using ABAC models. We have defined models, domains, data and security policies in OWL and used a reasoner to decide what is permitted. In this paper we present a way to represent the $ABAC_\alpha$ model using Web Ontology Language (OWL). The enforcement of policies is done using the EYE reasoner that infers the logical relationship and deduce the access grant for each requested action.**

***Keywords*—** Security Policy, ABAC, OWL, Reasoner, N3.

I. Introduction

Most organizations have policies that control their behavior. The ability to capture these policies, which are normally expressed in some natural language, in a machine understandable format has been an active thread of research (see for instance [3], [8], [10], [11]). The Web Ontology Language (OWL) [9] provides an efficient way to represent policies formally. Access control models when combined with formal policy specification language like OWL give the ability to write policies that describe entities and relationships in the system, how they affect access control, and how they are grounded out in models that are well understood in the security community. This combination further helps by using the power of reasoning to make access control decisions. A key contribution of our paper is to create an ontology and rules that capture the Attribute Based Access Control model, thus allowing for policies that are grounded out in ABAC. The specific model we capture is $ABAC_\alpha$ [4]. We have used the EYE [1] reasoner to infer more facts from the specified access control model, data and policies.

There are three successful access control models which are commonly used in practice: Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role Based Access Control (RBAC). However, these models have some shortcomings which limit their practical use. For example, in DAC it is possible that information may be accessed by unauthorized user due to the existence of multiple copies of the information and having no control on them. MAC avoids the problem by using the concept of security levels (*clearance* and *classification*). The security levels are applicable on each copy of the object. However, few security levels put a restriction on commercial applications. RBAC is based on user roles and associated permissions with each role. RBAC also suffers from problems like *role explosion* which corresponds to the situation in which different roles may require different permissions. This results in large number of roles and it becomes difficult to manage them.

ABAC model proposed by Jin *et. al* [4] is a general framework which combines the benefits of DAC, MAC and RBAC and goes beyond their limitations. The model is based on generic *attributes* which are used to capture identities and access control lists for DAC, clearances and classifications for MAC and roles for RBAC. This model allows for more flexibility for policy specifications as any number of *attributes* can be added within the same extensible framework. This also solves the shortcomings of the core RBAC model as appropriate attributes such as *location, business hours etc.* can be added within a unified framework. Dynamic computation of the authorization can be done at the time when the access request is made. We have used *Notation 3* (N3) [7] to express policies due to its more human readability.

In next section, we present the related work. Section III describes the $ABAC_\alpha$ model. Section IV describes the representation of the model in OWL. Finally we conclude with brief comment on the scope of the future work.

II. Related Work

Previous efforts were motivated by providing more flexibility in policy specifications and to enable automated inference of access control decisions. XACML [10] is a XML based general-purpose authorization policy model which enforces access control based on *attributes*. However it does not consider the semantics of the policies which poses a limitations on its use. In an other approach [5], XACML has been combined with OWL. There are other policy languages like Rei [11] which are based on deontic concepts. Rei associates access privileges with different credentials and properties of entities (user, agent, service, etc.). The language caters for the need of pervasive computing environment and provides flexible constructs like *policy objects*, *meta policies* and *speech acts* to express different types of policies. The Rein [8] framework builds on REI and is based on N3 rules. Rein

uses the cwm reasonng engine [12] to provide distributed reasoning capability over policy networks. In another work a multi-layer policy framework, KAoS [3] is proposed which supports policies described in OWL. Policy monitoring and enforcement is done automatically based on OWL ontologies.

ROWLBAC [6] was an effort to bring formalism into policy languages by modeling RBAC in OWL. In this representation entities (Users, Roles, Actions, etc.) are represented as OWL classes. Using OWL hierarchies and properties, different ontologies have been suggested. The proposed work suggests to go beyond RBAC and discusses about ABAC briefly.

The policy enforcement process infers additional facts about the data to determine information flow across the system. This necessitates a reasoning engine to be used. Euler Yet another proof Engine (EYE) [1] is an examples of a Prolog based reasoner. The reasoner runs in Prolog virtual machine and provides a generic reasoning environment by accepting its input in notation 3 (N3) rules. In this paper we build on ROWLBAC and use EYE reasoner to infer additional knowledge from the data.

III. ABAC$_\alpha$ Model

The overall structure of the ABAC$_\alpha$ model is shown in figure 1. The core component of the unified ABAC$_\alpha$ model are: users (U), subjects (S), objects (O), user attributes (UA), subject attributes (SA), object attributes (OA), permissions (P), authorization policies, and constraint checking policies for creating and modifying subject and object attributes. *Attributes* are functions which map an entity to a specific value from its range. While user *attributes* are created by security administrator, the *attributes* for subjects are created by users at the time of their creation. The same is true for object *attributes*. Permissions are privileges that a user can hold on objects and exercise via a subject. An authorization policy for a specific permission takes a subject, an object and returns true or false based on attribute values. The model also describes four policy configuration points which can be specified using a *Common Policy Language* (CPL). The first configuration point is for authorization policies. The second configuration point is constraints for subject attribute assignment. The third and fourth points are constraints for object attributes assignment at object creation and at object attribute modification respectively. Also, the *attributes* can be of two types: *set valued* and *atomic*. For example, *roles* associated with any user is a set attribute as the user may have different roles which can be activated based on some policy. On the other hand, *createdBy* is an atomic attribute which identifies the user who has created an object.

IV. Representation of ABAC$_\alpha$ Model in OWL

A. Basic Constructs of ABAC$_\alpha$

The basic constructs correspond to the various component of ABAC$_\alpha$ model mentioned in the previous section. These are represented as OWL classes using N3 as follows:

```
User a owl:class.   Subject a owl:class.
Object a owl:class. Permission a owl:class.
```

The *attributes* for users, subject and objects are defined as properties:

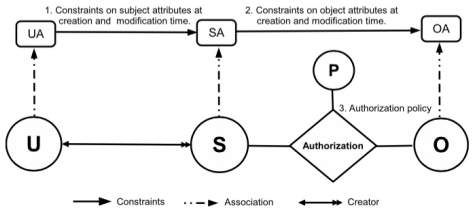

Fig. 1. Unified ABAC Model Structure as Proposed by Xin *et. al* [4]

```
UA a owl:ObjectProperty.
SA a owl:ObjectProperty.
OA a owl:ObjectProperty.
```

We also need to define a `RequestedAction` which corresponds to the request to access a specific object. In order to represent this we have imported the existing ROWLBAC [6] ontology (defined in `rbac` namespace) and extended its Action class. The `Action` class defined in ROWLBAC has exactly one subject and one object. We have extended it to create a `RequestedAction` class as follows:

```
RequestedAction a owl:class; rdfs:subClassOf Action.
permission a rdfs:Property, owl:FunctionalProperty;
  rdfs:domain rbac:Action;
  rdfs:range Permission.
```

Any individual who belongs to the `RequestedAction` class will have exactly one subject and it tries to acquire exactly one permission on exactly one object. Based on the inference process, it is decided if the `RequestedAction` is a `PermittedAction` or `ProhibitedAction`.

Each of the classical access control model is defined in a separate namespace. These namespaces are `abdac` (*attribute based* DAC), `abmac` (*attribute based* MAC) and `abrbac` (*attribute based* RBAC). The `data` namespace, defined for each model, contains sample instantiations.

B. Discretionary Access Control (DAC)

We define a property `subCreator` which binds the subjects to the user who created it. This is required as objects are accessed by subjects and we need to check if their creator is in the Access Control List (ACL) for the desired permission.

```
subCreator a owl:ObjectProperty;
  rdfs:domain Subject;  rdfs:range User.
```

The ACLs serves as the *attributes* for the objects. There is a one-to-one correspondence between permission and object attribute. For example, we may have `writer` attribute for `write` permission as follows:

```
writer a owl:ObjectProperty;
  rdfs:subPropertyOf abdac:OA;
  rdfs:domain abdac:Object; rdfs:range abdac:User.
```

The enforcement of DAC ensures that access is granted based on ACL *attributes*. For the policy *An object can be written only by subjects whose users are in the writer list of this object*, the access rule is specified as:

```
{ ?A a abdac:RequestedAction;
     abdac:subject ?S; abdac:object ?O;
     abdac:permission ?P.
  ?S abdac:subCreator ?U. ?O data:writer ?U.
  ?P rdfs:label "writeAccess"^^xsd:String.
} => { ?A a rbac:PermittedAction }.
```

The EYE [1] reasoner is used to decide this policy rule and can be invoked from command line as:

```
eye.sh rule.n3 data.n3 abdac.n3 --pass --nope >
output.n3
```

C. Mandatory Access Control (MAC)

In Mandatory Access Control (MAC), users and objects are assigned *clearance* and *classification* levels respectively. Access to a particular object is permitted only if some relationship exists between these two. The relationship is specified by some security model e.g. Bell LaPadula [14]. The security levels are defined using OWL classes in our ontology. The properties `clearanceLevel` and `classificationLevel` map the individuals of these classes to a numeric value for comparisons. For example, `Clearance` is defined as:

```
Clearance a owl:Class;
  rdfs:subClassOf
  [ a owl:Restriction;
      owl:onProperty clearanceLevel;
      owl:cardinality "1"^^xsd:nonNegativeInteger ].
```

In ABAC_α approach, subject clearance and object sensitivity serve as the *attributes* for subjects and objects. Properties `sclearance` and `sensitivity` bind subjects and objects to their security level. Classes for security levels are instantiated based on domain specific implementation requirements. For example, an organization may want to implement Bell LaPadula [14] model where subject clearance and object classification come from the same set {Top Secret, Secret, Confidential, Unrestricted}. As an example, `Top Secret` class is instantiated as:

```
TS a abmac:Clearance, abmac:Classification;
  abmac:clearanceLevel 3;
  abmac:classificationLevel 3.
```

The enforcement of policy is then straightforward. For example, the *read* policy rule as per the Bell LaPadula model [14] is: *A subject is allowed to acquire a read permission on a object if the object classification level is less than or equal to the subject clearance*. This is implemented as follows:

```
{ ?A a abmac:RequestedAction;
     abmac:subject ?S;  abmac:object ?O;
     abmac:permission ?P.
  ?S abmac:sclearance ?sc.
  ?sc abmac:clearanceLevel ?sl.
  ?O abmac:sensitivity ?oc.
  ?oc abmac:classificationLevel ?ol.
  ?P rdfs:label "readAccess"^^xsd:String.
  ?sl math:notLessThan ?ol.
} => { ?A a rbac:PermittedAction }.
```

D. Role Based Access Control (RBAC)

The ABAC_α model support only flat and hierarchical RBAC which are termed as RBAC_0 and RBAC_1 respectively. The model does not strive to provide other types of RBAC including dynamic separation of duties. We leverage the work done by Finin et al. [6].

We have considered a small portion of an academic scenario as an example domain for our ontology development. It is defined in the `acadDomain` ontology. There are different privileges associated with each role. For example, a user with `Faculty` role is allowed to perform `giveGrades` action which users with different roles may not be allowed to do. Similarly different printers may be accessed by different roles based on the place they are located in.

1) Roles as Attributes: In ABAC_α roles serve as *attributes* and access control is implemented based on the privileges associated with these attributes. We have defined roles for users, subjects and objects. As an example user role is defined as follows:

```
uRole a owl:ObjectProperty;
  rdfs:subPropertyOf UA;
  rdfs:domain User;
  rdfs:range rbac:Role.
```

Roles are created by defining individuals of the `Role` and its sub classes. For example, `PermanentFaculty` and `VisitingFaculty` roles are defined as:

```
PermanentFaculty a owl:Class;
  rdfs:subClassOf Faculty.
VisitingFaculty a owl:Class;
  rdfs:subClassOf Faculty.
```

At this point the static separation of duties can be provided using OWL's `disjointWith` feature as:

```
PermanentFaculty owl:disjointWith VisitingFaculty.
```

2) Enforcement of RBAC: For flat RBAC the access to a particular object is permitted if there exists *some* role attribute of the subject which is listed in the role attributes of the object for requested permission. The policy to enforce print access to a printer is defined as follows:

```
{ ?A a abrbac:RequestedAction;
     abrbac:subject ?S; abrbac:object ?O;
     abrbac:permission ?P.
  ?P rdfs:label "printAccess"^^xsd:String.
  ?S abrbac:srole ?r.
  ?O abrbac:orole ?r.
} => { ?A a abrbac:PermittedAction }.
```

In hierarchical RBAC, we also need to check if there exists *some* subject role which has a child-parent relationship with the object role attributes. For example, if a printer can be accessed by any `Faculty` then a subject with `PermanentFaculty` role must be allowed to access the printer as being `PermanentFaculty` implies `Faculty`. This is done as:

```
{ ?A a abrbac:RequestedAction;
     abrbac:subject ?S;  abrbac:object ?O;
     abrbac:permission ?P.
  ?P rdfs:label "printAccess"^^xsd:String.
  ?S abrbac:srole ?r1. ?O abrbac:orole ?r2.
  ?r1 rdfs:subClassOf ?r2.
} => { ?A a rbac:PermittedAction }.
```

E. Other Policy Configuration Points

1) Constraint for Subject Attribute at Creation time (`ConstrSub`): This configuration point constrains the attributes of the subjects at the time when they are created. We define `ConstrSubAction` as a class which has a user, a subject and one pair of attribute name and attribute value pair. The range of attribute value (`attrValueConstrSub`) is defined as the set of individuals of the desired attribute class. For DAC, we do not require this feature as the access grant is decided based on `SubCreator` attribute. For MAC, we can write the policy rule to ensure that the proposed clearance level of the subject is permitted only if it is same or less than the clearance level of the user as follows:

```
{ ?A a abmac:ConstrSubAction;
  abmac:userConstrSub ?U; abmac:subjectConstrSub ?S;
   abmac:attrNameConstrSub ?An;
   abmac:attrValueConstrSub ?Av.
  ?An string:matches "sclearance".
```

```
  ?U abmac:uclearance ?uc.
  ?uc abmac:clearanceLevel ?ul.
  ?Av abmac:clearanceLevel ?Avl.
  ?ul math:notLessThan ?Avl.
} => { ?A a abmac:PermittedConstrSub }.
```

Similar policy rules can be written for flat and hierarchical RBAC. In this case the range of `attrValueConstrSub` is `Role`. For flat RBAC, we check if the proposed subject role is an existing user role while for hierarchical RBAC, role inheritance is also checked in the same way as it was done in the case of authorization policy.

2) Constraint for Object Attribute at Creation time (`ConstrObj`): This configuration point constrains the attributes of the objects at the time when they are created. We define `ConstrObjAction` as a class which has a subject, an object and one pair of attribute name and attribute value pair. For DAC, apart from specifying members of the access control lists a subject can specify another attribute for the object such as `createdby`. However, the value of this attribute must be same as the creator of the subject. For MAC, a subject can create an object with the security level being same or higher than the security level of the subject. This is specified in a policy rule as:

```
{ ?A a abmac:ConstrObjAction;
   abmac:subjectConstrObj ?S;
   abmac:objectConstrObj ?O;
   abmac:attrNameConstrObj ?An;
   abmac:attrValueConstrObj ?Av.
  ?An string:matches "sensitivity".
  ?S abmac:sclearance ?sc.
  ?sc abmac:clearanceLevel ?sl.
  ?Avs abmac:classificationLevel ?Avl.
  ?Avl math:notLessThan ?sl.
} => { ?A a abmac:PermittedConstrObj }.
```

This feature is not application to RBAC as it does not talk about object attribute assignment by subjects.

3) Constraint for Object Attribute at Modification time (`ConstrObjMod`): This configuration point constrains the attributes of the objects at some point of time after their creation. We define `ConstrObjModAction` as a class which has a subject, an object and one pair of attribute name and attribute value pair. In DAC, the owner of the object can modify its access control lists. MAC (with tranquility) does not permit modification of an objects classification. $RBAC_0$ and $RBAC_1$ do not speak to this issue.

For example, in DAC, a policy rule to check if a reader can be added to the access control list of an object can be written as:

```
{ ?A a abdac:ConstrObjModAction;
     abdac:subjectConstrObjMod ?S;
     abdac:objectConstrObjMod ?O;
     abdac:attrNameConstrObjMod ?An;
     abdac:attrValueConstrObjMod ?Av.
  ?An string:matches "reader".
  ?S abdac:subCreator ?Sc. ?O abdac:createdby ?Sc.
} => { ?A a abdac:PermittedConstrObjMod }.
```

V. Conclusion

In this paper we have shown how the *Attribute Based Access Control* model can be represented using *Web Ontology Language* (OWL). This is a starting step towards formally specifying and enforcing machine understandable policies that can be captured in the ABAC model, which is one of the most general access control models available today. The unified $ABAC_\alpha$ model is proven to provide DAC, MAC and RBAC. We have written ontologies for each of these classical control models. The policy enforcement is shown using inference based reasoner EYE [1]. The performance of reasoning process is subject to further analysis. The basic $ABAC_\alpha$ model is not complete and does not cover static/dynamic separation of duties. It also lacks in subjects carrying additional attributes other than the corresponding users to reflect contextual information. All these features are represented in a more generic next level model: $ABAC_\beta$ [2]. In ongoing work we are modeling more complex policies by capturing the $ABAC_\beta$ model in OWL.

Acknowledgment

This work was partially done while author Joshi was on sabbatical at IIT Delhi. Support for his sabbatical visit from IIT Delhi is gratefully acknowledged. Support was also provided in part to Joshi by NSF awards *CNS1228673* and *IIS0910838*. Authors appreciate the discussions with Dr. G. Athithan, Ms. Anu Khosla, Prof. Tim Finin, and Dr. Lalana Kagal that helped in developing some of these approaches. Authors would also like to thank Dr. Kolin Paul and Dr. S.K. Gupta for their help.

References

[1] R. Verborgh, Jos De Roo, *Drawing Conclusions from Linked Data on the Web The EYE Reasoner*, IEEE Software, May-June 2015.

[2] Xin Jin, *Attribute-Based Access Control Models and Implementation in Cloud Infrastructure as a Service*, Phd Dissertation, The University of Texas, San Antonio, May 2014.

[3] J. Bradshaw, A. Uszok, M. Breedy, L. Bunch, T. Eskridge, P. Feltovich, M. Johnson, J. Lott, and M. Vignati, *The KAoS Policy Services Framework*, in Eighth Cyber Security and Information Intelligence Research Workshop (CSIIRW 2013), 2013.

[4] Xin Jin, Ram Krishnan and Ravi Sandhu, *A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC*, Proceedings of 26th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec 2012), Paris, France, July, 2012.

[5] Rodolfo Ferrini, Elisa Bertino, *Supporting RBAC with XACML+OWL*, SACMAT'09, Stresa, Italy, June 2009.

[6] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. H. Winsborough, and B. Thuraisingham. *ROWLBAC - Representing Role Based Access Control in OWL*, Proceedings of the 13th Symposium on Access control Models and Technologies, June 2008.

[7] T. Berners-Lee, D. Connolly, E. Prud?homeaux, Y. Scharf, *Experience with N3 rules*. In W3C Rules language Workshop, 2005.

[8] L. Kagal, T. Berners-Lee, *Rein: Where Policies Meet Rules in the Semantic Web*, Technical Report, MIT, 2005.

[9] M. Dean and G. Schreiber. *OWL Web Ontology Language Guide, 2004*. W3C Recommendation 2004, http://www.w3.org/TR/owl-guide/.

[10] S. Godik, T. Moses. *OASIS extensible access control markup language (XACML). OASIS Committee Specification cs-xacml-specification-1.0*, November 2002.

[11] L. Kagal, *Rei : A Policy Language for the Me-Centric Project* HP Labs, Tech. Rep., Sep. 2002.

[12] T. Berners Lee, *Cwm*, 2000.

[13] R. Sandhu, P. Samarati. *Access Control: Principals and Practice*, IEEE Communications Magazine, September 1994.

[14] D. E. Bell, L. J. LaPadula, *Secure Computer Systems: Mathematical Foundations*, vol. 1, 1973: MITRE Corp.