Modern Applied Science; Vol. 12, No. 10; 2018
ISSN 1913-1844 E-ISSN 1913-1852
Published by Canadian Center of Science and Education

An Algebraic Model to Analyze Role-Based Access Control Policies

Khair Eddin Sabri[1]

[1] Computer Science Department, King Abdullah II School of Information Technology, The University of Jordan, Amman, Jordan

Correspondence: Khair Eddin Sabri, Computer Science Department, King Abdullah II School of Information Technology, The University of Jordan, Amman 11942, Jordan. Tel: 962-6-5355000. E-mail: k.sabri@ju.edu.jo

Received: August 7, 2018 Accepted: September 6, 2018 Online Published: September 12, 2018
doi:10.5539/mas.v12n10p50 URL: https://doi.org/10.5539/mas.v12n10p50

Abstract

Role-Based Access Control (RBAC) is a well known access control model used to preserve the confidentiality of information by specifying the ability of users to access information based on their roles. Usually these policies would be manipulated by combining or comparing them especially when defined in a distributed way. Furthermore, these policies should satisfy predefined authorization constraints.

In this paper, we present an algebraic model for specifying and analyzing RBAC policies. The proposed model enables us to specify policies and verify the satisfaction of predefined authorization constraints. Furthermore, the model allows us to combine policies and analyze their effect on predefined constraints. The model consists of few operators that give simplicity in specifying polices. We present a prototype tool used for facilitating the analysis.

Keywords: information security, role-based access control policies, formal specification, algebraic analysis

1. Introduction

Many organizations require their information be stored securely so that it can be accessed only by legitimate users while others should be denied this access. One way to provide the confidentiality of information is to control its access and be limited to legitimate users only. Therefore, policies are specified to describe the allowed access for each user, and then mechanisms are implemented to enforce these policies. There are several access control approaches available in the literature to represent policies such as Discretionary Access Control (DAC) where a policy is determined by the owner of an object, Mandatory Access Control (MAC) which consists of rules that grant users access to resources, Role-Based Access Control (RBAC) that gives authorization to roles instead of individuals, and Attribute-Based Access Control (ABAC) which gives authorization based on the attributes of users instead of their identities.

RBAC is one of the most used approaches in organizations especially when the number of users is large. However, in such organizations, managing policies becomes complex. For example, an administrator may need to analyze the defined policies to ensure secure properties based on predefined constraints. A policy can be viewed as a rule which states a privilege of a user to access a specific object, whereas constraints are expressions defined to verify security properties. For example, a constraint can be stated as the developer cannot access the codes that are in quality assurance phase. Another constraint related to the hierarchy of rules could state that the division manager has more authority than the department manager. Therefore, any policy that gives an authorization to the department manager should give it as well to the division manager. There are different kinds of constraints presented in the literature such as constraints based on multi-level security as in (Bell & La Padula 1976) model. This model gives security labels to objects (e.g., information) and subjects (e.g., users). Each object and subject is assigned a security level. Bell-LaPadula model enforces its constraints by prohibiting any flow of information from a high level to a lower one. Other constraints are based on the Chinese wall model (Brewer & Nash 1989) that prohibits a user from accessing two sets of information that belong to the conflict set. However, these two models are restrictive i.e., do not give flexibility in defining policies and are not intended for RBAC.

Other models are presented in the literature that give flexibility in defining policies and enforcing predefined constraints such as (Ahn & Sandhu 2000), (Crampton 2003), (Jaeger & Tidswell 2001), (Sabri & Hiary 2016), and (Sabri 2015). However, these models do not allow calculation on policies, such as analyzing the effect of composing policies on the enforcement of constraints.

Managing policies, especially when defined from different resources, would require combining them to get a unified global policy, or comparing them to select the most restricted policy to apply. Several models are proposed in the literature to handle the composition of policies such as (Bonatti 2002) and (Wijesekera & Jajodia 2003) but not comparing them.

In this paper, we propose an algebraic model to analyze role-based access control policies. Our model is an information algebra (Kohlas & Stärk 2007). Information algebra is an abstract algebraic system that links several representations of information into one structure. Its main element is information which can be a policy. Each information is associated with a frame which can be seen as a classification to its parts e.g., role, object, subject, etc. Information algebra enables us through its operators to compare policies and combine them. The main contribution of this paper is proposing a comprehensive algebraic model to specify and analyze RBAC policies and enforce constraints.

The main features of the proposed algebraic model are:

- The model allows specifying and enforcing constraints.
- The model allows combining and comparing policies.
- The proposed algebraic model allows having calculation. For example, it allows analyzing the effect of composing information on the satisfaction of a constraint.
- The model is based on a well-defined theory called information algebra which enables us to inherits all its theories.
- The model consists of few operators which gives simplicity in specifying and analyzing policies.

The structure of the paper is as follows: Section 2 summarizes information algebra. Section 3 presents the proposed algebraic model and its application in analyzing policies. Section 4 demonstrates our prototype tool used in the analysis of access control policies. Section 5 presents related work. Finally, we conclude in Section 6.

2. Information Algebra

Kohlas and Stärk (Kohlas & Stärk 2007) introduced an abstract algebraic structure $((\Phi,\cdot),(D,\curlyvee,\curlywedge),\downarrow,d,e)$ to connect several representations of information such algebraic specifications, relational databases, modules, and constraint systems (Kohlas & Stärk 2007) and (Khedri, Chiang & Sabri 2013). Also, information algebra is used as an abstract structure to represent knowledge of agents in analyzing the flow of information in multi-agent system (Sabri, Khedri & Jaskolka 2009b). Information algebra contains a set of information Φ and an operator $\cdot$ to represent combining information. Furthermore, it contains a lattice of frames $(D,\curlyvee,\curlywedge)$ that gives a classification to the information with an ordering relation $\preccurlyeq$. In addition, information algebra contains an operator $\downarrow$ to restrict an information to a part that belongs to a specific frame, and an operator d that gives the frame of an information. Each frame x contains an information called the empty information e_x. Information algebra is any algebraic system that satisfies the following ten axioms.

(1) $(\varphi\psi)\chi = \varphi(\psi\chi)$

(2) $\varphi\psi = \psi\varphi$

(3) $d(\varphi\psi) = d(\varphi) \curlyvee d(\psi)$

(4) $x \preccurlyeq y \Rightarrow (e_y)^{\downarrow x} = e_x$

(5) $d(\varphi) = x \Rightarrow \varphi e_x = \varphi$

(6) $\forall(x \mid x \in D : d(e_x) = x)$

(7) $x \preccurlyeq d(\varphi) \Rightarrow d\left(\varphi^{\downarrow x}\right) = x$

(8) $x \preccurlyeq y \preccurlyeq d(\varphi) \Rightarrow (\varphi^{\downarrow y})^{\downarrow x} = \varphi^{\downarrow x}$

(9) $d(\varphi) = x \wedge d(\psi) = y \Rightarrow (\varphi\psi)^{\downarrow x} = \varphi(\psi^{\downarrow x \curlywedge y})$

(10) $x \preccurlyeq d(\varphi) \Rightarrow \varphi\varphi^{\downarrow x} = \varphi$

Axioms 1 and 2 state that combining information is commutative and associative. Axiom 3 relates combining pieces of information to their frames. Axioms 4, 5, and 6 show properties of the empty information. Axioms 7 and 8 present properties of the restricting operator. Axioms 9 and 10 show properties of the combining and restricting operators together.

3. The Proposed Algebraic Model

In this section, we propose an algebraic model to represent role based access control and show that our model is information algebra, and therefore, reuse the theory of information algebra. In our model, a RBAC authorization policy is an information. We represent information as a function formulated as a set of 2-tuples (i,A) where i is an index used to classify an information and A is a set such as $\{(\mathrm{i}, \mathrm{A}), \mathrm{i} \in \mathrm{J} \wedge \mathrm{A} \in \mathbb{A}_{\mathrm{i}}\}$, where $\mathbb{A}_{\mathrm{i}}$ is the set of all elements that are classified as *i* and *J* is a frame. We use the set Φ to represent the set of all authorization policies that can be defined in an organization, and use the set Φ_p to denote the defined authorization policies in an organization.

The set of all possible classifications of information is the set of frames and is denoted by I such that $i \in I$. The lattice D constructed from the frames is $(\wp(I),\cup,\cap)$. In RBAC policies, the set of frames is I = {role, object, privilege} because a RBAC policy consists of a role, object, and a privilege. An empty information is defined as $e \triangleq \{(i,\emptyset) \mid i \in J\}$, where $J \in D$ is a frame. We call the structure $\mathcal{N} = (\Phi_p, (\Phi, D)$ a policy structure.

Definition 1 (RBAC Authorization Policy). Let φ be an information such that $\varphi \in \Phi$. φ is an authorization RBAC policy iff $d(\varphi) = \{role, object, privilage\}$.

Definition 1 requires that an authorization policy should include information about role, object and privilege. For example, the information {(role, {manager}), (object, {file_1}), (privilege, {read})} is an authorization policy that states any user playing the role of manager can read from file_1. A policy in our structure states the allowed actions of subjects on resources. It does not include the prohibited actions.

A policy that contains an empty set is called partial policy such as {(role, {manager}), (object, {file_1}), (privilege, {}). In this paper, when we use the term policy we mean non-partial policy unless otherwise stated. The information {(role, {Alice}), (object, {file_1})} is not an authorization policy as the domain of the information is not *I*.

In addition to the knowledge of policies, RBAC contains knowledge that links roles to subjects $\mathcal{N}^r = (\Phi_p^r, (\Phi^r, D^r)$. For example, an information in that knowledge can be {(role, {Manager}), (subject, {Alice})} which indicates that Alice plays the role of manager. The frame of information in $\mathcal{N}^r$ is $d(\varphi) = \{role, subject\}$. Later in this section, we show the way of relating privileges to subjects based on their roles. The next section discusses the different ways to combine policies and to compare them based on their flexibility.

3.1 Manipulating Policies

We present two ways of combining policies. The first one expands the authorization. For example, an organization may specify that the manager is allowed to read from file_1, and another organization specifies that the manager is allowed to write into file_1. By combining the two authorization policies, we get the policy {(role, {manager}), (object, {file_1}), (privilege, {read, write})} that allows the manager to read and write into file_1. We define this way of combining policies as follows:

$$\varphi \cdot \psi \triangleq \{(i,A) \mid i \in I \wedge A = \varphi(i) \cup \psi(i)\}$$

The domain of φ and ψ is the same which is {role, object, privilege}. Usually, we represent the combining operator $\varphi \cdot \psi$ as $\varphi\psi$ (omit the dot).

Another way of combining policies is to restrict the privileges of the two policies by considering the common ones only. For example, by combining the authorization policy {(role, (manager}), (object, {file_1, file_2}), (privilege, {read})} with the policy {(role, {manager}), (object, {file_1}), (privilege, {read, write})} we get a more restricted policy that allows the manager to read from file_1 {(role, {manager}), (object, {file_1}), (privilege, {read})}. We define this operator as:

$$\varphi * \psi \triangleq \{(i,A) \mid i \in I \wedge A = \varphi(i) \cap \psi(i)\}$$

In case no common privilege exists in φ and ψ, combining them $\varphi * \psi$ gives a partial policy.

To focus and extract part of the policy based on a frame, we define the following binary operator $\uparrow: \Phi \times D \to \Phi$

$$\varphi^{\uparrow J} \triangleq I_J ; \varphi$$

where $J \in D$ and $\varphi \in \Phi$. For example, let φ = {(role, {manager}), (object, {file_1}), (privilege, {read})} and *J* = {role}. Then $\varphi^{\uparrow J}$ extracts the role of the policy which is the manager $\varphi^{\uparrow J}$ = {(role,{manager})}.

It is proved in (Sabri, Khedri & Jaskolka 2009a) that $(\Phi,\cdot)$ is an idempotent commutative semigroup. Therefore, it has a natural order $\leq$ such that $\varphi \leq \psi \iff \varphi \cdot \psi = \varphi$. We use this relation to compare policies based on their flexibility.

Definition 2 (More restricted relation). An authorization policy φ is more restricted than the authorization policy ψ iff all the privileges provided by φ are also provided by ψ. We denote that by $\varphi \leq \psi$.

For example, let φ and ψ be two authorization policies such that φ = {(role, {manager}), (object, {file_1}), (privilege, {read})} and ψ = {(role, {manager, developer}), (object, {file_1}), (privilege, {read})}. Here, φ is more restricted than ψ as ψ allows both developers and managers to read from file_1 while φ allows only manager. Formally, $\varphi \leq \psi$ because $\varphi \cdot \psi = \psi$.

3.2 Composite Policy

An information can be a composite policy. For example, the information {(subject, {Bob}), (object, {file_2}), (privilege, {read, write})} is a composite policy that contains two authorizations. The first one indicates that Bob can read from file_2. The second one states that Bob can write into file_2.

Definition 3 (Elementary Policy). Let φ, ψ, χ be policies such that $\psi \neq \chi$. We call φ an elementary policy iff $\forall(\psi, \chi \mid \psi, \chi \in \Phi : \psi\chi \neq \varphi)$.

In other words, a policy is elementary if the set associated with each classification is singleton. We use the predicate elementary(φ) to represent elementary policy. From a composite policy, we can extract all its elementary policies using the singleton function defined as

$$singleton(\varphi) = \{ \psi \mid elemantry(\psi) \wedge \psi \leq \varphi\}$$

For example, let the policy φ = {(role, {manager}), (object, {file_2}), (privilege, {read, write})}. Then, singleton(φ) = { {(role, {manager}), (object, file_2}), (privilege, {read})} , {(role, {manager}), (object, {file_2}) , (privilege, {write})} }.

3.3 Set of Policies

Usually, a set of authorization policies are defined to control the access of objects in an organization. These policies state the privileges of roles on objects. This is what we denote by Φ_p. These policies can be either elementary or composite. For example, the following policies can be defined in an organization to state that the manager is allowed to write into file_1, while the developer is allowed to write into file_1 and read and write into file_2.

(1) {(role, {manager}), (object, {file_1}), (privilege, {write})}.

(2) {(role, {developer}), (object, {file_1}), (privilege, {write})}.

(3) {(role, {developer}), (object, {file_2}), (privilege, {read, write})}.

Based on the operators of our model, we define two functions on the set of information. The first function is *isThereAPolicy* defined formally as $isThereAPolicy(\mathcal{N}, \varphi) \triangleq \exists(\psi \mid \psi \in \Phi : \varphi \leq \psi)$. This function checks the existence of a policy ψ in the policy structure $\mathcal{N} = (\Phi_p, (\Phi, D))$ that is more flexible than φ. In other words, this function checks the existence of a policy, within the defined authorization policies in an organization, that allows the request ψ. The set Φ is the set of all possible policies, and Φ_P is the set of defined policies in an organization. The second function extracts all policies satisfying a specific condition. The function is defined as $extract(\mathcal{N}, J, \varphi) \triangleq \{\psi^{\downarrow J} \mid J \in D \wedge \varphi \leq \psi \wedge J \subseteq d(\psi)\}$ which returns pieces of information from the policy structure $\mathcal{N}$ that contain φ after restricting them to the frame J.

We use these two functions to check the ability of a user to access an object. The user request consists of a subject, role and the requested action. We represent the access request as an information. Before authorizing a subject to access an object, the following steps should be performed

- extract the role of the requested subject $\varphi_r \in extract\left(\mathcal{N}^r, \{Role\}, \varphi_a^{\downarrow\{subject\}}\right)$.
- combine the role with the object and the requested privilege $\varphi = \varphi_r \cdot \varphi_a^{\downarrow\{subject, privilage\}}$.
- verify the existence of a policy that gives an authorization to the requested access through the function $isThereAPolicy(\mathcal{N}, \varphi)$.

Furthermore, we can compare two set of polices based on their flexibility. To perform the comparison, we apply the function Singleton defined as $Singleton(\Phi) = \{ \varphi \mid \psi \in \Phi \wedge \varphi \in singleton(\psi)\}$. Then, use set operators to compare the two sets of policies.

3.4 Constraints Validation

In this section we present different kinds of constraints and show their representations within our algebraic model.

Hierarchy: An organization can state a constraint that the manager role should have more authority than the nurse role. To verify the compliance of a new defined policy in the organization with that constraint, we can perform the following check. Assume that φ is the policy to be defined in the organization. Let ψ = {(role, {manager}) and χ = {(role, {nurse}). The verification can be represented as $\psi \leq \varphi \Rightarrow \chi \leq \varphi$ which indicates that giving a nurse a privilege implies that the manager is given the same privilege.

Cardinality: An organization can state a constraint that no subject is assigned to more than one role. We can verify this constraint using the proposed algebraic model based on the $\mathcal{N}^r$ knowledge since it contains the association between roles and subjects. In order to perform the verification, we first assume that all the pieces of information in Φ_p^r are elementary which can be achieved by using the Singleton function. Then, we extract all the subjects from $\mathcal{N}^r$ as $S = extract(\mathcal{N}^r, \{subject\}, e_\emptyset)$. This function extracts all the available subjects i.e., having the empty information in the condition of the extract function. After that, we extract the set of roles for each subject and verify that the extracted set contains one role as $\forall(\varphi \mid \varphi \in S : R = extract(\mathcal{N}^r, \{role\}, \varphi) \wedge |R| = 1)$.

Another example of the cardinality is for example the total number of roles that can read from file_1 should be at least three. This constraint can be verified within our model as $extract(\mathcal{N}, \{role\}, \varphi) > 3$ where $\varphi = \{(\text{subject}, \{\text{file\_1}\}), (\text{privilege}, \{\text{read}\})\}$.

The verification is based on the extract function which depends on the relation $\varphi \le \psi$ where φ is a condition and ψ is an organization policy. It has been proved in (Sabri & Hiary 2017) that $\varphi \le \psi \Rightarrow \varphi \le \psi\chi$. Therefore, combining the policies of the organization ψ with other policies would not affect the satisfaction of the constraint ($\varphi \le \psi$), and therefore, does not require rechecking. Furthermore, restricting the condition would not affect the constraint as well $\varphi \le \psi \Rightarrow \varphi * \chi \le \psi$.

Separation of duties: An organization may state that two particular roles should never be assigned to the same person such as no user should play the role of developer and QA. This constraint can be verified by extracting all the subjects playing the role of developer and playing the role of QA. The intersection should be the empty set.

$$extract(\mathcal{N}^r, \{subject\}, \{(role, \{developer\})\}) \cap extract(\mathcal{N}^r, \{subject\}, \{(role, \{QA\})\}) = \emptyset$$

By using the proposition $\varphi \nleq \psi \Rightarrow \varphi \nleq \psi * \chi$, we can conclude that restricting the policies in the organization would not affect this constraint and therefore, rechecking is not required.

4. Tool

We develop a prototype tool that helps in defining and manipulating RBAC policies. This tool is built on the top of the tool developed by (Sabri el. al. 2009a). Our tool enables us to test if an information is a policy as:

```
isPolicy ([("object",["file1"]),("privilege",["write"]), ("role",["manager"])], ["object","privilege","role"])

result: True

isPolicy ([("object",["file1"]),("role",["manager"])], ["object","role"])

result: False
```

Also, the tool allows checking if a policy is an elementary and to extract all the elementary policies from a composite one.

```
isSinglePolicy ([("object",["file1"]),("privilege",["write"]),("role",["manager"])],
                ["object","privilege","role"])

result: True

singleton ([("object",["file1","file2"]),("privilege",["write"]),("role",["manager"])],
                ["object","privilege","role"])

result:

[([("object",["file1"]),("privilege",["write"]),("role",["manager"])],
                ["object","privilege","role"]),

([("object",["file2"]),("privilege",["write"]),("role",["manager"])],
                ["object","privilege","role"])]
```

We implement the two operators that combine policies

```
combinePolicies1

([("object",["file1"]),("privilege",["write"]), ("role",["manager"])],
                ["object","privilege","role"])

([("object",["file2"]),("privilege",["write"]),("role",["manager"])],
                ["object","privilege","role"])

result:
```

```
([("object",["file1","file2"]),("privilege",["write"]),("role",["manager"])],
                 ["object","privilege","role"])
combinePolicies2
([("object",["file1"]),("privilege",["read","write"]),("role",["manager"])],
                 ["object","privilege","role"])
  ([("object",["file1"]),("privilege",["write"]),("role",["manager","developer"])],
                 ["object","privilege","role"])
result:
([("object",["file1"]),("privilege",["write"]),("role",["manager"])],
                  ["object","privilege","manager"])
```

We can compare the flexibility of two policies by using the policyLess function.

```
policyLess
([("object",["file1"]),("privilege",["write"]),("role",["manager"])],
                  ["object","privilege","role"])
([("object",["file1"]),("privilege",["write"]),("role",["manager","developer"])],
                  ["object","privilege","role"])
result: True
```

We implement the function Singleton() that returns all the elementary policies from a set of policies.

```
singletonSet
[ ([("object",["file1","file2"]), ("privilege",["write"]),("role",["manager"])],
                ["object","privilege","role"]),
([("object",["file1"]), ("privilege",["write"]),("role",["manager","developer"])],
                ["object","privilege","role"])
]
result:
[ ([("object",["file1"]),("privilege",["write"]),("role",["manager"])],
                ["object","privilege","role"]),
([("object",["file1"]),("privilege",["write"]),("subject",["developer"])],
                ["object","privilege","role"]),
([("object",["file2"]),("privilege",["write"]),("subject",["manager"])],
                ["object","privilege","role"])
]
```

We also implement functions that represent the two operators of combining set of policies. Also, we implement a function that compares two sets of policies, and a function that checks their equality.

```
setPoliciesEqual
[ ([("object",["file1","file2"]), ("privilege",["write"]),("role",["manager"])],
                ["object","privilege","role"]),
([("object",["file1"]), ("privilege",["write"]),("subject",["manager","developer"])],
                ["object","privilege","role"])
]
[ ([("object",["file1"]), ("privilege",["write"]),("role",["manager","developer"])],
                ["object","privilege","role"]),
([("object",["file2"]), ("privilege",["write"]),("subject",["manager"])],
        ["object","privilege","role"])
]
result: True
```

5. Related Work and Discussion

Several methods are available in the literature for specifying and analyzing access control policies. In this section, we summarize the most related models and compare them with the proposed one. (Fisler, Krishnamurthi, Meyerovich, & Tschantz,2005) present a formal way to verify policies written in xacml language related to known properties with the use of the software suite Margrave. An example of a property given in (Fisler et al. 2005) is that there is no student who can assign external grades. Also, an example of a policy is given as: Requests for students to receive external grades. We can represent this policy as {(role, {student}), (object, {ExternalGrades}), (privilege, {receive})} and perform the verification as described in the paper. (Crampton 2003) develops a set-based specification scheme for authorization constraints in role-based access control systems and suggests an enforcement model to satisfy these constraints. (Ahn & Sandhu 2000) introduce a formal language for specifying role-based authorization called RCL 2000. They apply the language to express authorization constraints such as separation of duties. Our model allows us, in addition to this kind of verification, to compose and compare policies.

(Adi, Bouzida, Hattak, Logrippo, & Mankovskii 2009) use context expressions for specifying policies at different levels of abstraction, and define a type system to detect conflicts. (Bruns, Dantas, & Huth 2007) define an access control policy language as a four-valued predicate. The language is based on Belnap logic and the truth values corresponds to "grant", "deny", "conflict", and "undefined". This language is used in (Bruns & Huth 2008) for the analysis and composition of policies. Constraint logic programming is used by (Barker & Stuckey 2007) to represent an extension to the RBAC model which allows defining access policies that may include features, like denials of access and temporal authorizations. Furthermore, (Sabri & Obeid 2016) develop a theory based on defeasible logic to handle conflicting policies. However, this language is not used for specifying and enforcing constraints.

Also, algebraic methods are introduced in the literature for the analysis of access control policies. For example, the paper of (Bonatti, De Capitani di Vimercati, & Samarati 2002) proposes an algebra for composing policies. It provides an algebraic method for composing them such as union, intersection and difference. This paper does not provide a way of comparing the flexibility of policies. (Li & Wang 2006) introduce an algebra for the specification of high level policies that combine requirements of users' attributes with requirements of the number of users. Also, they study the potential mechanisms to enforce these policies and the complexity related to policy analysis and enforcement. However, the focus of this paper is on the separation of duties policies which is different from ours. (Wijesekera & Jajodia 2003) present a propositional algebra to model access control policy operators such as union, intersection, sequential composition, etc. However, the access control model of this paper is not RBAC.

6. Conclusion

In this paper, we present an algebraic model to specify and analyze Role-based Access Control (RBAC) authorization policies. We give a formal definition to a policy, and we distinguish between elementary and composite policies. Furthermore, we define operators on policies and set of policies. The main advantages of the structure is that it enables us to analyze RBAC policies at two different levels: policy level and set of policies level. Also, the presented model enables us to manipulate policies and sets of policies by combining and comparing them, check authorization constraints, and control user requests to access resources.

We developed a prototype tool implemented using Haskell programming language for the specification and manipulating of policies. As a future work we aim at extending the model to involve the set of prohibited objects from a set of user subjects and to detect potential conflicts as a result of combining policies.

References

Adi, K., Bouzida, Y., Hattak, I., Logrippo, L., & Mankovskii, S. (2009). *Typing for conflict detection in access control policies*, E-Technologies: Innovation in an Open World, 4th International Conference, MCETECH 2009, Ottawa, Canada, May 4-6. Proceedings (G. Babin, P. G. Kropf, and M. Weiss, eds.), 26 of Lecture Notes in Business Information Processing, Springer. https://doi.org/10.1007/978-3-642-01187-0_17

Ahn, G. J., & Sandhu, R. (2000). Role-based authorization constraints specification. *ACM Transactions on Information and System Security, 3*, 207-226. https://doi.org/10.1145/382912.382913

Barker, S., & Stuckey, P. J. (2003). Flexible access control policy specification with constraint logic programming, *ACM Transactions on Information and System Security, 6*, 501-546. https://doi.org/10.1145/950191.950194

Bell, D., & La Padula, L. (1976). Secure computer system: Unified exposition and multics interpretation, *Tech. Rep. ESD-TR-75-306*, The MITRE Corporation, March 1976.

Bonatti, P., De Capitani di Vimercati, S., & Samarati, P. (2002). An algebra for composing access control policies, *ACM Transactions on Information and System Security*, 5, 1-35. https://doi.org/10.1145/504909.504910

Brewer, D. F., & Nash, M. J. (1989). *The Chinese wall security policy*. IEEE Symposium on Security and Privacy, pp. 206-214. https://doi.org/10.1109/SECPRI.1989.36295

Bruns, G., Dantas, D. S., & Huth, M. (2007). *A simple and expressive semantic framework for policy composition in access control*. Proceedings of the 2007 ACM workshop on Formal methods in security engineering, FMSE '07, (New York, NY, USA), pp. 12-21, ACM. https://doi.org/10.1145/1314436.1314439

Bruns, G., & Huth, M. (2008). *Access-control policies via belnap logic: Effective and efficient composition and analysis*. Proceedings of the 2008 21st IEEE Computer Security Foundations Symposium, (Washington, DC, USA), pp. 163-176, IEEE Computer Society. https://doi.org/10.1109/CSF.2008.10

Crampton, J. (2003). *Specifying and enforcing constraints in role-based access control*, Proceedings of the eighth ACM symposium on Access control models and technologies, SACMAT '03, (New York, NY, USA), pp. 43-50, ACM. https://doi.org/10.1145/775412.775419

Fisler, K., Krishnamurthi, S., Meyerovich, L. A., & Tschantz, M. C. (2005). *Verification and change-impact analysis of access-control policies*. Proceedings of the 27th international conference on Software engineering, ICSE '05, (New York, NY, USA), pp. 196-205. https://doi.org/10.1145/1062455.1062502

Jaeger, T., & Tidswell, J. E. (2001). Practical safety in flexible access control models, *ACM Transactions on Information and System Security, 4*, 158-190. https://doi.org/10.1145/501963.501966

Khedri, R., Chiang, F., & Sabri, K. E. (2013). *An algebraic approach towards data cleaning*, Proceedings of The 4th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2013), vol. 21 of Procedia Computer Science, 50-59. https://doi.org/10.1016/j.procs.2013.09.009

Kohlas, J., & Stärk, R. F. (2007). Information algebras and consequence operators, *Logica Universalis, 1*, 139-165. https://doi.org/10.1007/s11787-006-0007-2

Li, N., & Wang, Q. (2006). *Beyond separation of duty: An algebra for specifying highlevel security policies*. Proceedings of the 13th ACM conference on Computer and communications security, CCS '06, (New York, NY, USA), 356-369, ACM. https://doi.org/10.1145/1379759.1379760

Sabri, K. E. (2015). Automated Verification Of Role-Based Access Control Policies Constraints Using Prover9, The *International Journal of Security, Privacy and Trust Management, 4*(1), 1-10. https://doi.org/10.5121/ijsptm.2015.4101

Sabri, K. E., & Hiary, H. (2016). Algebraic Model for Handling Access Control Policy. *Procedia Computer Science, 83*, 653-657. https://doi.org/10.1016/j.procs.2016.04.146

Sabri, K. E., Khedri, R., & Jaskolka, J. (2009a). Specification of agent explicit knowledge in cryptographic protocols, *International Journal of Electrical and Computer Engineering, 4*(2), 122-129.

Sabri, K. E., Khedri, R., & Jaskolka, J. (2009b). *Verification of information flow in agent based systems*, Proceedings of the 4th MCETECH Conference on e-Technologies (G. Babin, P. Kropf, and M. Weiss, eds.), vol. 26 of Lecture Notes in Business Information Processing, 252-266, Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/978-3-642-01187-0_22

Sabri, K. E., & Obeid, N. (2016). A temporal defeasible logic for handling access control policies. *Applied Intelligence, 44*(1), 30-42. https://doi.org/10.1007/s10489-015-0692-8

Wijesekera, D. & Jajodia, S. (2003). A propositional policy algebra for access control, *ACM Transactions on Information and System Security, 6*, 286-325. https://doi.org/10.1145/762476.762481