MORoles: An Abstract Hierarchy Model for Managing Overlapping Security Roles

Ahmad Mousa Altamimi†
Applied Science Private University, Amman, Jordan

Summary
Protecting data against unauthorized access is an essential demand for any information system. Such protection ranges from simple authentication to the very complex authorization while at the same time ensuring accessibility to authorized users. To achieve these security considerations, security policies are defined, usually by the system administrator, for controlling and monitoring users accessing. These policies consist of a series of constraints associated with a set of roles that, in turn, may be assigned to one or more users according to their duties. In most cases, user's roles do not overlap or conflict. However, in a rapidly changing systems, a user would likely have more than one role, and some of these roles may very well overlap. In this paper, an abstract hierarchy nature security model (MORoles) that has been specifically designed for managing overlapping security roles is presented. MORoles ensures that security roles are mutually consistent by organizing roles into a hierarchy structure to support a more expressive representation and then extracts the highest non-conflicting roles amongst the user's assigned roles. To underscore the practical visibility of the proposed approach, the open source library tree.hh is utilized to provide a practical implementation.

Key words:
Security Policies, Privacy, Overlapping, Roles Hierarchy.

1. Introduction

Considerable effort has been devoted to formally defining security policies in information systems. In fact, three levels of policy specification having been identified in the literature [1]. The first one is the High-level abstract policies, which can be business goals, service level agreements, or trust relationships. These policies are not enforceable and their realization involves refining them into one of the other two policy levels. The second level is the Specification-level policies or business-level policies, which are specified by the system administrator and related to specific objects. And finally, the Low-level policies or configurations such as security mechanism configurations or device configurations that are related to hardware. In this research, we focus on the second kind of policies (Specification policies) and discuss the concepts used to express these policies. After that we present a well-structured model that is designed to manage the overlapping policies roles.

Specification policies (hereafter simply referred to as "policies") determine which user, under what circumstances, may access specific information. This can be accomplished by defining a series of conditions (constraints), usually by the system administrator, for controlling and monitoring user access [2]. In general, a policy is determined by the sensitivity of the information. If it is sensitive, a policy should be developed to maintain tight control over accessing that info. For instance, within a hospital the pathological history of patients may be considered as sensitive data. The policy could establish that only doctors and nurse practitioners may access the pathological history of patients; any other user should be restricted from accessing this data. In fact, in alike systems, security considerations range from simple policies to the very complex policies in order to secure the sensitive information.

That is, supporting such policies are typically based upon an integration of three basic elements. Users to which authorizations are granted. A User can be single or a group of users within the system. Data to be protected, which can be any part of the stored information. And finally, Roles, which are named collections of authorizations or privileges granted to Users to perform certain job functions. For example, let us assume that we have an organization in which roles are created based on the job functions of users. Roles are subsequently have a set of Constraints based on the requirements of the roles' jobs. Users in turn are then assigned appropriate roles based on their qualification.

In most cases, a user can be authorized to play several roles, with overlapping permissions. These roles may be organized into a hierarchy to support a more expressive representation of their semantics. So before enforcing policies, any conflicting authorizations may need to be resolved. In other words, a user with two different roles may have privileges to access all parts of information authorized for one roles but restricted by the other roles. In this case, the user should be given the highest permissions amongst his/her roles by finding the highest non-conflicting roles.

To address such issue, our model organizes the roles into a hierarchical structure (i.e., tree structure) using an open source library called tree.hh [3], then the inheritance is resolved in the context of overlapping roles and finally the non-conflicting permissions among the assigned roles are extracted. More discussions about the model is provided in section 5.

Manuscript received July 5, 2017
Manuscript revised July 20, 2017

The rest of this paper is organized as follow: Section2 discusses the preliminaries terminologies relevant to this approach. Section 3 in turn, presents an overview of related work. Our methodology is discussed in Section 4. The proposed model itself is then presented in Section 5 along with its implementation in Section 6. Final conclusions are offered in Section 7.

2. Preliminaries

Before discussing our model, we provide a brief overview of the basic terminology and structures relevant to policies in general.

2.1 Users, Objects and Roles

In fact, policies consist of roles, data/objects, and users. Roles are first created with specific constraints or authorization on particular objects (data) based on the job functions. Users in turn, would be a user or even a program permitted to perform particular operations based on their assigned roles. The using of such elements is particularly useful for common operations such as adding/dropping a user, or assigning/changing user roles. Figure 1 depicts the relationship.

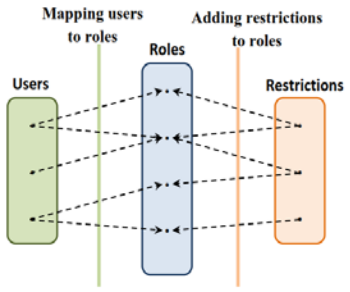


Fig. 1 Users, Roles, and Restrictions relationship

The consolidation of access control for many users into a single role entry allows for much easier management of the overall system and much more effective verification of security policies. However, in large systems, role hierarchy - and the need for finer-grained customized privileges - makes administration potentially unwieldy. As such, the management of an individual's constraints becomes much simpler in that constraints do not have to be directly assigned on a user-by-user basis. Further details about roles hierarchy are given next.

2.2 Roles Hierarchy

Roles may have organized into a hierarchy which, in turn, defines a partial ordering, denoted as $\preccurlyeq$ [4]. Such hierarchies support a more expressive representation especially in the case of existing overlapping permissions. More formally, we can say that given a role domain R, let r_i , r_j $\in R$ be individual roles. If r_i precedes r_j in the hierarchy ordering ($r_i \preccurlyeq r_j$), we say that r_i is partially ordered relative to r_j and, furthermore, that r_i is a child of r_j, and r_j is a parent of r_i. This implies that r_i inherits all constraints that are assigned to r_j, and that all users who are mapped to r_i are affected by the r_j constraints. This is formally expressed in *Definition 1.*

Definition1: A role r_i in a role hierarchy H inherits all constraints of roles R = (r_j , …, r_z), where $r_i \preccurlyeq r_j$ and $r_j \preccurlyeq r_x \preccurlyeq r_z$ for all roles $r_x \in R$. We say that r_i inherits all constraints of roles reachable from r_i to the Root role of R.

An example of a role hierarchy is illustrated in Figure 2, where any role inherits all constraints that are assigned to its parents up to the Root role. For instance, suppose that a Store table with four attributes (e.g., Country, Province, City, and Store_Number) should not be accessed by users of the Marketing role. Consequently, any user who is assigned to the Marketing Role or any of its children is restricted from accessing the Store table and, by extension, is also restricted from accessing all the attributes of the specified table. This what will be called Security Object and defined formally in Definition 2.

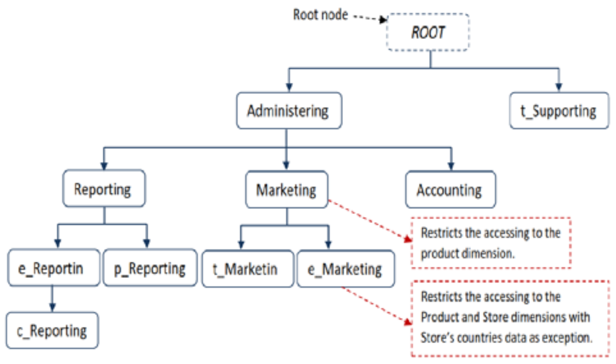

Root node
ROOT
Administering
t_Supporting
Reporting
Marketing
Accounting
Restricts the accessing to the product dimension.
e_Reportin
p_Reporting
t_Marketin
e_Marketing
Restricts the accessing to the Product and Store dimensions with Store's countries data as exception.
c_Reporting

Fig. 2 Roles hierarchy

Definition 2. A Security Object O for roles R = {{v}: {v}= values of tables T restricted to roles R, where {v} = all T's attributes vales}.

3. Related Work

Policies have been extensively considered in the literature. For example, many languages have been designed or extended for expressing policies such as the XML concepts based languages [5, 6, 7] and the logic programming based languages [8, 9, 10]. One can consider for example the eXtensible Access Control Markup

Language (XACML), which is one of the most relevant proposed languages [11, 5]. Such XML-based languages are particularly suitable to convey requirements related to authorization and privacy for web-based systems [12].
Other researchers considered the policies formulation. Authors of [13, 14], for instance, utilized the Unified Modeling Language (UML) to express trust policies using predicate expressions whose grammar is expressed in UML. Similarly, authors of [15] presented a Trust Management Framework that supports policy life cycle management using UML diagrams.
Policy specification in web-based applications has been also proposed in [16, 17, 18]. SELinks, for instance, targets web apps and provides a uniform programming model (in the style of LINQ and Ruby on Rails), with language syntax for accessing objects residing either in the database or at the server [17]. Still other frameworks investigate the association of security policies with client side code, with protection provided by the interception and analysis of database queries [19]. For the most part, however, none of these works are designed to manage or consider the overlapping roles but the policy design itself.
Recently, the object-oriented paradigm has been utilized to provide a policy specification model [20]. In that research, the features set of the object-oriented paradigm (i.e., the concepts of classes and objects) was borrowed to create instances of various policy constructs such as Users, Roles, and Objects. These instances are then combined together to create more expressive policies. A primary objective of this approach is to allow policy designers to identify security constructs at the level of the conceptual data model, without regard for the complexity of the underlying logical or physical implementation.

4. The Methodology

Managing overlapping policies requires a formal basis (i.e., a suitable model with clear architecture) to ensure that they are mutually consistent and to allow the using of policies without requiring modifications to the existing access control mechanism. For this purpose, we propose a well-structured model that is hierarchy based nature. To this ends, roles are organized into a hierarchy to support a more expressive representation. Then the highest permissions amongst the assigned roles are extracted by finding the highest non-conflicting roles.
To address such issue, our model relies on an open source library called tree.hh to organize the roles into a hierarchical structure (i.e., tree structure), then the inheritance is resolved in the context of overlapping roles and finally the highest non-conflicting permission are found. A detailed discussion for our model is provided in the next section.

5. The MORoles Model for Managing Overlapping Security Roles

As illustrated, it would be helpful to be able to assign a user to more than one role according to his/her duties. Each role would have different permissions to access specific information. In most cases, roles do not overlap/conflict. However, in a rapidly changing enterprise environment, a user would likely have more than one role, and some of these roles may very well overlap. In fact, most existing information systems apply a “Restriction takes Precedence” principle. However, the problem with this approach is that it may lead to unintended restrictions on accessible data. In other words, if a user is a member of several roles with different permissions, the user will be restricted according to the permissions of the least powerful role, which results in restricting the user from access to data even if he/she is permitted to access them by using other role(s).
For example, suppose the administrator of an information system is included in the roles shown in Figure 2 and assigns user Alice to the Administration role, which has full access to the whole information. Over time, and because of special situations, Alice is assigned also to the Marketing role, which restricts her from accessing Product data. In this example, Alice's roles conflict; Alice is restricted from accessing Product data because of the Marketing role, but at the same time, she is allowed to access the same data because of the Administration role.
To address this issue, Alice should be given the highest permissions amongst her roles by finding her highest non-conflicting roles. To do this, roles should be organized in a hierarchal structure (i.e., a tree structure) in order to pick-up the highest role/node. That is, within a role-hierarchy restrictions will be inherited such that a role may inherit all restrictions that are assigned to its parents up to the top of the hierarchy. Consequently, all users who are mapped to this role are affected by the role's restrictions plus all the inherited restrictions as formally expressed in Definition 3.

Definition 3: A User u who assigned to roles R_n = (r_1 … r_n) affected by all constraints of R_n PLUS all constraints of R_n hierarchies H_n = (h_1 … h_n), where h_1 is the hierarchy for r_1 … and h_n is the hierarchy for r_1 , and h_x consists of r_j , …, r_z, where $r_i \preccurlyeq r_j$ and $r_j \preccurlyeq r_x \preccurlyeq r_z$ for all roles $r_x \in R$. We say that u restricts by all constraints of R_n PLUS all constraints reachable from r_i to the Root role of R.

To ground our conceptual work, in the next section, we discuss the implementation of such a hierarchy and we also give an example to illustrate both the power and intuitive nature of our approach.

6. MORoles Model Implementation

To represent roles in a hierarchal structure (i.e., a tree structure), we employ the open source tree.hh library. Tree library is an STL-like container class designed to represent n-ary trees [3]. It provides various types of iterators such as breadth first, depth first, and sibling iterators to traverse the tree nodes where its access methods are compatible with the C++ STL libraries. In our case, we assume a depth first search strategy, where the algorithm traverses the tree starting from the root role(s) and explores as far as possible along each branch before backtracking. If the role is identified as an assigned role, the role's restrictions are extracted and the traversal then backtracks to another branch. In the worst case, we have to visit each node exactly once, since we do not cross the same edge more than once. As such, the time complexity is O(n) where n is the number of roles/nodes, which is generally quite small.

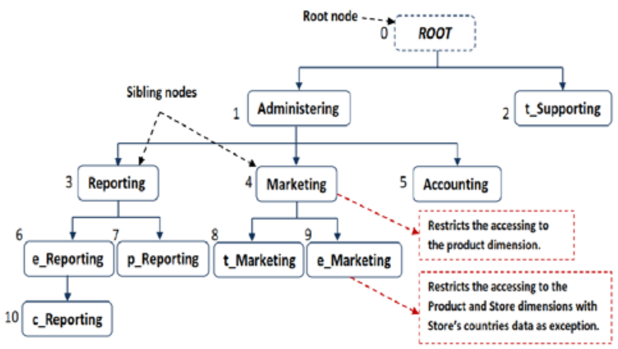
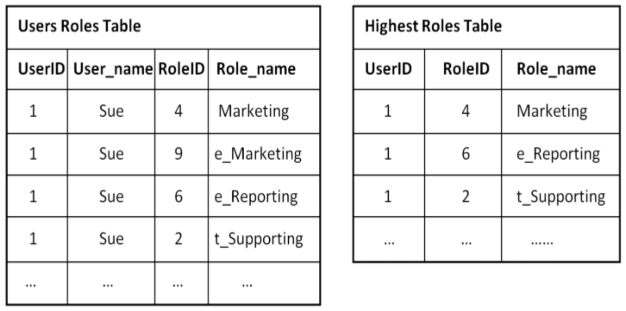

Root node
0
ROOT
Sibling nodes
1
Administering
2
t_Supporting
3
Reporting
4
Marketing
5
Accounting
Restricts the accessing to the product dimension.
6
7
8
9
e_Reporting
p_Reporting
t_Marketing
e_Marketing
Restricts the accessing to the Product and Store dimensions with Store's countries data as exception.
10
c_Reporting

Fig. 3 Roles Hierarchy represented in a tree structure

Figure 3 illustrates the Role Tree associated with the roles hierarchy depicted in Figure 2. Numbers near each node represent the roleID. In the roles tree, every node is connected to an arbitrary number of child nodes/roles. At the top of the tree, there may also exist a set of roles which are characterized by the fact that they do not have any parents. Nodes at the same level are called “siblings”' and are not overlapping. So, if a user is assigned to sibling roles, the user's permissions will be the union of all his role's restrictions. However, nodes at different levels may indeed overlap. Each node may inherit its parent's restrictions if any exist.
To improve the search performance, the user's role(s), along with their restrictions, are stored in a relational repository. The highest or most privileged roles amongst the user roles are also stored there. So, instead of re-executing the search process each time the user request a n access to the stored information, his/her highest roles are retrieved from the repository and cached in memory for future accessing. Figure 4 illustrates an example for a user’s roles stored in the repository.



Users Roles Table

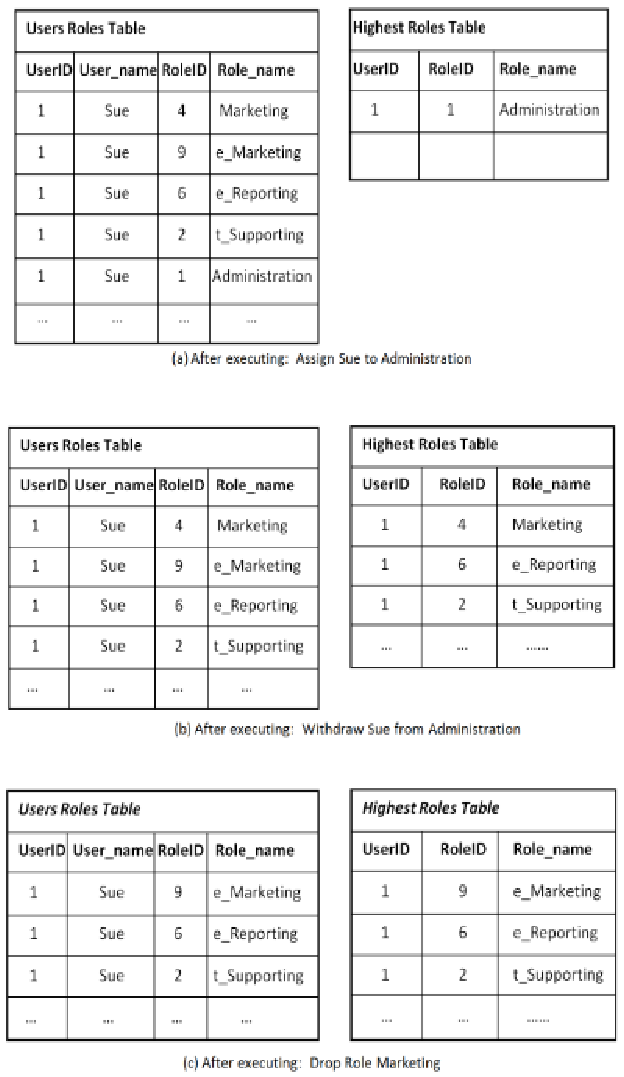
UserID	User_name	RoleID	Role_name
1	Sue	4	Marketing
1	Sue	9	e_Marketing
1	Sue	6	e_Reporting
1	Sue	2	t_Supporting
...	...	...	...

Highest Roles Table

UserID	RoleID	Role_name
1	4	Marketing
1	6	e_Reporting
1	2	t_Supporting
...	...	......

Fig. 4 User’s roles in the Policy Repository

Of course, the user's roles can be changed or affected over the time (e.g., Assign, Withdraw, and Drop roles). For example, assume that the user Sue is assigned to the following roles: Marketing, e_Marketing, e_Reporting, and t_Supporting. Sue's restrictions will be defined by the union of these roles. Note, e_Marketing is a child of the Marketing role and because the user utilizes the highest role, e_Marketing is not listed in the highest roles table. As a consequence, its restrictions will be ignored.
Now, suppose the user Sue becomes an Administrator user. The roles Marketing, t_Marketing, and e_Reporting will no longer be listed in the highest roles table, and their restrictions will be ignored in the security checking process because they are children of the Administration role. Finally, suppose that the policy is once again altered and the user Sue is withdrawn from the same Administration role. The user's highest roles should then be reset to Marketing, e_Reporting, and t_Supporting. Figure 5 shows an example for the Assign, Withdraw and Drop operations, and their effect on the security tables.
Ultimately, we note that the Roles Tree itself may also be affected by the Drop operation. For instance, when a role is dropped, the tree is re-structured by moving all children of the deleted role so that they become siblings of that role. For example, suppose the Marketing role is dropped. Here, the t_Marketing and e_Marketing roles should be connected directly to the Administration role. For this purpose, we also provide a simple algorithm to rebuild the tree. It starts from the parent of the deleted node, extracts the sub-trees of its child nodes, and then attaches each one to the parent of the deleted role. The full tree can be re-structured if necessary. Figure 6 shows the roles tree after dropping the Marketing role.



Users Roles Table

UserID	User_name	RoleID	Role_name
1	Sue	4	Marketing
1	Sue	9	e_Marketing
1	Sue	6	e_Reporting
1	Sue	2	t_Supporting
1	Sue	1	Administration
...	...	...	...

Highest Roles Table

UserID	RoleID	Role_name
1	1	Administration

(a) After executing: Assign Sue to Administration

Users Roles Table

UserID	User_name	RoleID	Role_name
1	Sue	4	Marketing
1	Sue	9	e_Marketing
1	Sue	6	e_Reporting
1	Sue	2	t_Supporting
...	...	...	...

Highest Roles Table

UserID	RoleID	Role_name
1	4	Marketing
1	6	e_Reporting
1	2	t_Supporting
...	...	......

(b) After executing: Withdraw Sue from Administration

Users Roles Table

UserID	User_name	RoleID	Role_name
1	Sue	9	e_Marketing
1	Sue	6	e_Reporting
1	Sue	2	t_Supporting
...	...	...	...

Highest Roles Table

UserID	RoleID	Role_name
1	9	e_Marketing
1	6	e_Reporting
1	2	t_Supporting
...	...	......

(c) After executing: Drop Role Marketing

Fig. 5 How the Repository is affected by Assign, Withdraw, and Drop operations

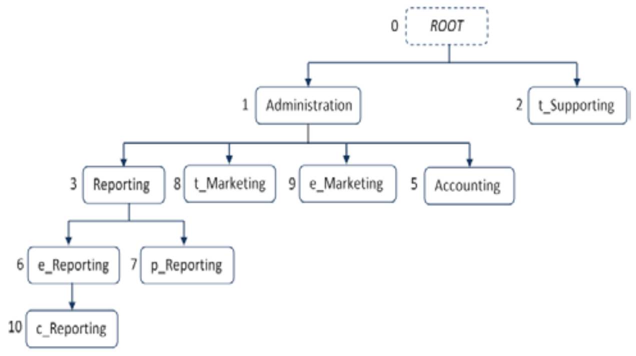


Fig. 6 The updated roles tree

7. Conclusions

In this paper, we have introduced a model that is designed specifically to manage overlapping security roles (MORoles). In most complex information systems, a user would likely have more than one role, and some of these roles may have conflicted constraints. MORoles is built upon the well-known tree hierarchy structure. As a consequence of organizing the roles in such a way, constraints of multi-level hierarchies would be inherited by the descendant roles. Moreover, security administrators not only work in a familiar setting, but would enable verifying security policies to ensure that they are mutually consistent

We have also discussed the implementation of MORoles using the open source STL-like container class library (tree.hh) that is designed to represent n-ary trees. Finally, a simple case study is carried out to discuss how the roles tree is affected/re-structured by applying different operations such as drop, assign, and revoke roles. In conclusion, we believe that the model presented in this work represents a significant contribution to the literature in that it gives a general solution to the problem of overlapping roles.

Acknowledgments

The authors are grateful to the Applied Science Private University, Amman-Jordan, for the full financial support granted to cover the publication fee of this research article.

References

[1] Nicodemos Damianou. A Policy Framework for Management of Distributed Systems. PhD thesis, Imperial College London, March 2002.

[2] Dieter Gollmann. Computer Security, 3rd ed. Wiley Publishing, 2011.

[3] tree.hh: an STL-like C++ tree class, June 2017. http://www.tree.phi-sci.com.

[4] Altamimi, Ahmad and Eavis, Todd: "Securing Access to Data in Business Intelligence Domains". The International Journal on Advances in Security, vol 5, no 3 & 4. pp. 94 – 111. 2012.

[5] OASIS. eXtensible Access Control Markup Language (XACML) Version 3.0, June 2017. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[6] Mariemma Yag¨ue. Survey on xml-based policy languages for open environments. Journal of Information Assurance and Security, 1(1):11–20, 2006.

[7] Kevin P. Twidle, Naranker Dulay, Emil Lupu, and Morris Sloman. Ponder2: A policy system for autonomous pervasive environments. In Radu Calinescu, Fidel Liberal, Mauricio Marn, Lourdes Pealver Herrero, Carlos Turro, and Manuela Popescu, editors, ICAS, pages 330–335. IEEE Computer Society, 2009.

[8] Moritz Y. Becker, C´edric Fournet, and Andrew D. Gordon. Secpal: Design and semantics of a decentralized authorization language. J. Comput. Secur., 18(4):619–665, December 2010.
[9] Matteo Dell'Amico, Gabriel Serme, Muhammad Sabir Idrees, Anderson Santana de Olivera, and Yves Roudier. Hipolds: a security policy language for distributed systems. In Proceedings of the 6th IFIP WG 11.2 international conference on Information Security Theory and Practice: security, privacy and trust in computing systems and ambient intelligent ecosystems, WISTP'12, pages 97–112, Berlin, Heidelberg, 2012. Springer-Verlag.
[10] Aarthi Nagarajan, Vijay Varadharajan, and Michael Hitchens. Alopa: Authorization logic for property attestation in trusted platforms. In Proceedings of the 6th International Conference on Autonomic and Trusted Computing, ATC '09, pages 134–148, Berlin, Heidelberg, 2009. Springer-Verlag.
[11] Sonia Jahid, Carl A. Gunter, Imranul Hoque, and Hamed Okhravi. Myabdac: compiling XACML policies for attribute-based database access control. In Proceedings of the first ACM conference on Data and application security and privacy, CODASPY '11, pages 97–108, New York, NY, USA, 2011. ACM.
[12] Q.Z. Sheng, Jian Yu, Z. Maamar, Wei Jiang, and Xitong Li. Compatibility checking of heterogeneous web service policies using vdm++. In Services - I, 2009 World Conference on, pages 821–828, 2009.
[13] Masoom Alam, Ruth Breu, and Michael Hafner. Model-driven security engineering for trust management in sectet. Journal of Software, 2(1):4759, 2007.
[14] Muhammad Alam, Michael Hafner, Ruth Breu, and Stefan Unterthiner. A framework for modeling restricted delegation in service oriented architecture. Trust and Privacy in Digital Business, pages 142–151, 2006.
[15] Skogsrud Halvard, R. Motahari-Nezhad Hamid, Benatallah Boualem, and Casati Fabio. Modeling trust negotiation for web services. Computer, 42(2): 54–61, 2009.
[16] PieroAndrea Bonatti, JuriLuca Coi, Daniel Olmedilla, and Luigi Sauro. Rulebased policy representations and reasoning. In Franois Bry and Jan Mauszyski, editors, Semantic Techniques for the Web, volume 5500 of Lecture Notes in Computer Science, pages 201–232. Springer Berlin Heidelberg, 2009.
[17] Brian J. Corcoran, Nikhil Swamy, and Michael Hicks. Cross-tier, label-based security enforcement for web applications. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, SIGMOD '09, pages 269–282, New York, NY, USA, 2009. ACM.
[18] Ian Jacobi, Lalana Kagal, and Ankesh Khandelwal. Rule-based trust assessment on the semantic web. In Nick Bassiliades, Guido Governatori, and Adrian Paschke, editors, Rule-Based Reasoning, Programming, and Applications, volume 6826 of Lecture Notes in Computer Science, pages 227–241. Springer Berlin Heidelberg, 2011.
[19] Adrienne Porter Felt, Matthew Finifter, Joel Weinberger, and David Wagner. Diesel: applying privilege separation to database access. In Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS '11, pages 416–422, New York, USA, 2011. ACM.
[20] Altamimi, Ahmad and Eavis, Todd: OSSM: An Object Oriented Security Specification Model for OLAP Systems. The 25th Australasian Database Conference (ADC), Brisbane, Australia, 2014.

Ahmad Mousa Altamimi is assistance professor of computer science at Applied Science Private University. He has been received his PhD degree in Computer Science from Concordia University – Montreal, Canada in 2014 and his MSc degree in 2007. Altamimi has joined Applied Science University - the Faculty of Information Technology as an assistant professor since July 2014. His research interests are primarily in software engineering and data privacy for interactive and non-interactive database environments. Dr. Altamimi participated in the organization of many conferences, he was the publicity chair of CSIT 2016 conference.