Advanced Analysis of the Integrity of Access Control Policies: the Specific Case of Databases

Faouzi Jaidi, Faten Ayachi, and Adel Bouhoula
Digital Security Research Lab, Higher School of Communication of Tunis, University of Carthage, Tunisia

Abstract: Databases are considered as one of the most compromised assets according to 2014-2016 Verizon Data Breach Reports. The reason is that databases are at the heart of Information Systems (IS) and store confidential business or private records. Ensuring the integrity of sensitive records is highly required and even vital in critical systems (e-health, clouds, e-government, big data, e-commerce, etc.). The access control is a key mechanism for ensuring the integrity and preserving the privacy in large scale and critical infrastructures. Nonetheless, excessive, unused and abused access privileges are identified as most critical threats in the top ten database security threats according to 2013-2015 Imperva Application Defense Center reports. To address this issue, we focus in this paper on the analysis of the integrity of access control policies within relational databases. We propose a rigorous and complete solution to help security architects verifying the correspondence between the security planning and its concrete implementation. We define a formal framework for detecting non-compliance anomalies in concrete Role Based Access Control (RBAC) policies. We rely on an example to illustrate the relevance of our contribution.

Keywords: Access Control, Databases Security, Formal Validation, Integrity Analysis, Conformity Verification.

Received November 11, 2016; accepted July 7, 2019
https://doi.org/10.34028/iajit/17/5/14

1. Introduction
Securing a critical Information Systems (IS) requires basically setting up a trusted and reliable access control policy. Nonetheless, setting up a trustworthiness environment of access control and monitoring its compliance have emerged as complicated tasks. However, mastering these tasks is crucial to ensure a higher protection of IS.

As part of this thematic, we focus on the analysis of the integrity of concrete RBAC [21] policies within relational Data Base Management Systems (DBMS). The DBMS context allows a complete study of this problematic for two main reasons. First, a DBMS represents the heart of the IS. It acts as a firewall to control accesses to data, but unlike firewalls, the policy is managed in the same place and way as the data it protects and, as a consequence, it is highly exposed to corruption attempts. Second, it is commonly agreed that the policy is subject to different updates. So, it may face several deficiencies during its life-cycle:

1. It can record non-compliant changes regarding its original specification.
2. It may contain incoherent rules.
3. It is highly exposed to inner threats.

We propose a complete formal solution for the analysis of the reliability of low-level access control policies.

The remainder of this paper is structured as follows. In section 2, we discuss related works. In section 3, we focus on the problem of non-compliance of access control policies. In section 4, we technically define our formal verification, validation and analysis framework.

In section 5, we rely on an example to illustrate the relevance of our proposal. Finally, in section 6 we conclude the paper and introduce ongoing works.

2. Related Works
Numerous research works had treated the verification of the specifications of access control policies. In [4], the authors proposed to specify the policy via the an extension of Unified Modeling Language Secure (UML) [18] and to verify the obtained models by using the MDel, Measuring and VAlidating UML Class Diagrams tool Secure (MOVA) tool. This tool helps to evaluate the security model through Object Constraint Language (OCL) requests. To formally verify the specified policy, Idani et al. [9] proposed to encode the models specified with Secure UML in the Z language and to analyze the policy via the Jaza tool that allows animating the specifications. In [17], authors chose to specify access control policies via Secure UML and to transform the specifications to the B notation by using the B4M secure tool. The verification of the formalized policies is based on the animator Pro B tool. Koch et al. [15] chose to organize the set of roles in a graph which captures different variants of RBAC models. This formalism for structuring roles can take advantage from well-established results in graph transformation systems [20] and the issues addressed in [3, 19].

Ahmed and Arputharaj [2] proposed to map XACML policies and rules into relational rules, stored in tables within relational databases, to control access of XML document.
Several other works focused on the validation of access control policies. Researchers opted for representing roles in different concepts allowing the analysis, validation or optimization of the policy. Contributions deal with the following aspects:

1. Validating the implemented policy regarding the defined security constraints [7] by using a finite model checker.

In [6], the author chose to model the policy as a graph of roles and proposed two methods to use this graph: the first one is based on algorithms of the theory of graphs to follow the paths of the graph to find illicit transfer of privileges; the second one consists in storing the graph in a Lightweight Directory Access Protocol (LDAP) directory and developing a new LDAP schema to represent the graph of roles. In [22], the authors proposed a logical framework for enforcing the integrity of access control policies in the context of relational databases. They focused primarily on how to enforce and check constraints in concrete policies.

Existing research works deal with the verification of a specified RBAC policy to check its exactitude; or the validation of an implemented policy to make sure the correctness of its implementation regarding the defined security constraints. Checking the compliance between high and low levels of a policy (the policy planning and its implementation) according to our knowledge is not treated as much as necessary and needs more attention.

3. Synthesis of the Non-Compliance Problem

In most DBMSs, application roles are implicitly activated or settled during a user session often without restrictions. A malicious authorized user can take benefits to expand his scope of actions. A particular crucial problem is related to malicious administrators. If administrative roles are not used wisely, a malicious administrator can corrupt the policy and create security breaches such as the following scenarios [11]:

1. Users and Privileges Alteration: a malicious administrator may corrupt the policy via creating, removing or renaming users (resp. roles), assigning new privileges bypassing the original specification to avoid an audit or a system investigation.
2. Transmission of Access Rights: granting the privilege “create any role” or granting roles with the privilege “with admin option” allows the guarantees to delegate those roles and therefore generating a new access flow invisible from outside the database.
3. Alteration of the Access Flow: a malicious user who disposes of sufficient privileges to do so may corrupt the authorized access flow via altering the set of predefined privileges and may falsify the global behavior of the access control process.

4. Problems related to Roles Management: managing the hierarchy of application roles is not easy due to: no restrictions to control roles empowerment; and roles visibility has not been treated by security modeling languages. Hence, management tasks may generate conflicting roles difficult to identify.

5. Combining Different Mechanisms: the coexistence of access control mechanisms may prompt malicious administrators to assign rights in different models. This may generate security holes difficult to identify. For instance, direct assignment of actions to users is correct in the Discretionary Access Control (DAC) model [16], but is an offense in the RBAC model.

6. Violation of Implicit Negative Authorizations: most DBMS authorization models are based on the closed world policy: without authorization, the access is denied. Nevertheless, this approach doesn’t prevent a user from receiving the authorization some times in the future. So, it is difficult to verify if a specified negative authorization is still enforced.

4. Compliance Analysis of RBAC-Policies

Protecting a database from insider threats requires basically building profiles of normal accesses and identifying anomalous accesses with respect to those profiles [5]. Our reasoning to address this problematic offers a global vision of the process of developing trusted policies [10, 12]. It allows verifying and validating that a concrete policy instantiates well a valid specification model. Our approach consists of the following basic phases. Phase 1 concerns the specification of the policy based on SecureUML as a modeling language. Phase 2 concerns the encoding of the specified models in the B notation [1] based on an adjusted and adopted version of B4Msecure tool. Phase 3 defines reverse engineering techniques to extract the implemented policy from the DBMS. Phase 4 concerns the formalization of the extracted policy in the target notation. Phase 5 consists to formally verify and validate the conformity of the concrete policy regarding its specification.

4.1. Verification, Validation and Analysis Framework

We note ACP=(_USERS, ROLES, PERMISSIONS, AUR, ARR, APR) the formal representation of the specified policy with USERS is the set of users, ROLES is the set of roles, OBJECTS is the set of resources, ACTIONS is the set of access modes and:

- PERMISSIONS: is the set of permissions (actions on objects): \( \text{PERMISSIONS} \subseteq \text{ACTIONS} \times \text{OBJECTS} \).
4.1. Verification of Access Control Policies

To formally verify the exactitude of the concrete policy, we check that the generated B machines are coherent, well structured, syntactically and semantically correct. By using the Atelier B tool, we perform a number of demonstrations to verify the establishment of the invariants on the initialization and during operations calls. This tool allows checking types, generating and demonstrating proof obligations, etc. To verify the specified policy, we proceed in the same manner and in addition we use the Pro B tool which is an animator of specifications and a model checker to check the correctness of the specifications.

4.1.1. Conformity Validation of Concrete Policies

1. Definitions:

- Definition 1 [Power of a role]: we define the power of a role in terms of authorizations allocated to it.

- Definition 2 [Power of a user]: we define the power of a user either in terms of authorizations or in terms of roles allocated to it.

- Definition 3 [Permissions Of Role]: we note Permissions Of Role(\(r\)) the function defined in (1), that returns for each role \(r \in \text{ROLES}_{\text{IMP}}\) the set of permissions assigned to it.

\[
\text{PermissionsOfRole}(r) = \left\{ \begin{array}{ll}
(p_i) & \in \text{PERMISSIONS}_{\text{IMP}} | r \in \text{ROLES}_{\text{IMP}} \\
& r \in \text{ROLES}_{\text{IMP}} \land (r, p_i) \in \text{APR}_{\text{IMP}} \\
& (p_i) \in \text{PERMISSIONS} \\
r \in \text{ROLES} \land (r, p_i) \in \text{APR}.
\end{array} \right.
\]  

(1)

- Definition 4 [Roles Of User]: we note RolesOfUser(\(u\)) the function defined in (2) that returns for each user \(u \in \text{USERS}_{\text{IMP}}\) the set of roles assigned to it.

\[
\text{RolesOfUser}(u) = \left\{ \begin{array}{ll}
[r_i] & \in \text{ROLES}_{\text{IMP}} | u \in \text{USERS}_{\text{IMP}} \land (u, r_i) \in \text{ARR}_{\text{IMP}} \\
& (r_i) \in \text{ROLES} | u \in \text{USERS} \land (u, r_i) \in \text{AUR}.
\end{array} \right.
\]  

(2)

- Definition 5 [Permissions Of User]: we note Permissions Of User(\(u\)) the function defined in (3) that returns for each user \(u \in \text{USERS}_{\text{IMP}}\) the set of permissions indirectly assigned (via roles) to that user.

\[
\text{PermissionsOfUser}(u) = \{ u \in \text{rolesOfUser}(u) \mid \text{PermissionsOfRole}(r_i) \}. 
\]  

(3)

2. Anomalies Detection.

- Detecting Hidden Users: hidden users are new users (not initially defined) injected in the concrete policy. This is perceptible when \(\text{USERS}_{\text{IMP}} \neq \text{USERS} \). We compute the set of Hidden Users in (4) as the difference between the sets of implemented and specified users.

\[
\text{HiddenUsers} = \text{USERS}_{\text{IMP}} \setminus \text{USERS}.
\]  

(4)

- Detecting Hidden Roles: hidden roles are new roles (not initially planned) introduced in the concrete policy. This is observable when \(\text{ROLES}_{\text{IMP}} \neq \text{ROLES} \). We compute the set of Hidden Roles in (5) as the difference between the sets of implemented and specified roles.

\[
\text{HiddenRoles} = \text{ROLES}_{\text{IMP}} \setminus \text{ROLES}.
\]  

(5)

- Detecting Hidden Access Flow: hidden access flow belongs to Hidden Assignments of Roles to Roles (HARR), Roles to Users (HAUR) and Permissions to Roles (HAPR).

\[
\text{HAUR} = \text{AUR}_{\text{IMP}} \setminus \text{AUR}.
\]  

(6)

\[
\text{HARR} = \text{ARR}_{\text{IMP}} \setminus \text{ARR}.
\]  

(7)

\[
\text{HAPR} = \text{APR}_{\text{IMP}} \setminus \text{APR}.
\]  

(8)

We compute the hidden access flow (HiddenACFlow) in (9) as the union of the sets of hidden assignments. Generally, the union operator requires the same typing for all the sets to be combined. Since types checking are already done in the verification phase, we consider Hidden AC Flow as the union of enumerated sets.

\[
\text{HiddenACFlow} = \text{HAUR} \cup \text{HARR} \cup \text{HAPR}.
\]  

(9)

- Detecting Missed Users: missed users are initially specified users but not defined in the concrete policy. This is identified...
when \( \text{USERS} - \text{USERS}_{\text{IMP}} \neq \emptyset \). In (10), we calculate the set of Missed Users as the difference between the sets of specified and implemented users.

\[
\text{MissedUsers} = \text{USERS} - \text{USERS}_{\text{IMP}}. 
\]

- Detecting Missed Roles: missed roles are initially planned roles but not implemented or removed. This is observable when \( \text{ROLES} - \text{ROLES}_{\text{IMP}} \neq \emptyset \). We compute the set of Missed Roles in (11) as the difference between the sets of specified and implemented roles.

\[
\text{MissedRoles} = \text{ROLES} - \text{ROLES}_{\text{IMP}}.
\]

- Detecting Missed Access Flow: missed access flow concerns Missed Assignments of Roles to Roles (MARR), Roles to Users (MAUR) and Permissions to Roles (MAPR).

\[
\text{MAUR} = \text{AUR} - \text{AUR}_{\text{IMP}}
\]

MAUR: logically, this case is detectable when \( \text{AUR} - \text{AUR}_{\text{IMP}} \neq \emptyset \). In (12), we compute MAUR as the difference between the two sets of specified and implemented assignments of users to roles.

\[
\text{MARR} = \text{ARR} - \text{ARR}_{\text{IMP}}
\]

MAUR: logically, this situation is observable when \( \text{ARR} - \text{ARR}_{\text{IMP}} \neq \emptyset \). We compute MARR in (13) as the difference between the two sets of specified and implemented assignments of roles to roles.

\[
\text{MAPR} = \text{APR} - \text{APR}_{\text{IMP}}
\]

- Detecting Renamed Users: it belongs to users whose names have been changed but still dispose of the same privileges. We identify this when: \( \exists \, \tau_i \in \text{MissedRoles}, \tau_j \in \text{HiddenRoles} \mid \exists \, p = (o, a), ((\tau_i, o, a) \in \text{APR} \land (\tau_j, o, a) \in \text{APR}_{\text{IMP}}) \lor ((\tau_i, o, a) \in \text{APR} \land (\tau_j, o, a) \in \text{APR}_{\text{IMP}}) \), and we compute it in (17) as the set of couples of hidden and missed roles sharing the same permissions.

\[
\text{RenamedUsers} = \{(u, r) \mid u \in \text{MissedUsers} \land \exists \, \tau_i \in \text{HiddenUsers} \land \left( \text{PermissionsOfUser}(u) = \text{PermissionsOfUser}(\tau_i) \right) \land \left( \text{RolesOfUser}(u) = \text{RolesOfUser}(\tau_i) \right)\}
\]

- Detecting Renamed Roles: it regroups roles whose names have been changed but still dispose of the same privileges. We identify this when: \( \exists \, \tau_i \in \text{MissedRoles}, \tau_j \in \text{HiddenRoles} \mid \exists \, p = (o, a), ((\tau_i, o, a) \in \text{APR} \land (\tau_j, o, a) \in \text{APR}_{\text{IMP}}) \lor ((\tau_i, o, a) \in \text{APR} \land (\tau_j, o, a) \in \text{APR}_{\text{IMP}}) \), and we compute it in (17) as the set of couples of hidden and missed roles sharing the same permissions.

\[
\text{RenamedRoles} = \{(u, r) \mid u \in \text{MissedRoles} \land \exists \, \tau_j \in \text{HiddenRoles} \land \left( \text{PermissionsOfRole}(\tau_i) = \text{PermissionsOfRole}(\tau_j) \right)\}
\]

4.1.3. Formal Analysis

Correctness and completeness proofs.

To prove the correctness of our reasoning, we consider the following cases of conformity analysis.
Let us denote by $\mathcal{A}$ the set of users and $\mathcal{R}$ the set of roles. The set of assignments is then $\mathcal{AUR} = \mathcal{A} \times \mathcal{R}$.

We define the following operators on sets of assignments:

- $\cap$ denotes the intersection,
- $\cup$ denotes the union,
- $\setminus$ denotes the difference.

We also define the following operators on sets of users:

- $\cap$ denotes the intersection,
- $\cup$ denotes the union,
- $\setminus$ denotes the difference.

We define the following operators on sets of roles:

- $\cap$ denotes the intersection,
- $\cup$ denotes the union,
- $\setminus$ denotes the difference.

We define the following operators on sets of users:

- $\cap$ denotes the intersection,
- $\cup$ denotes the union,
- $\setminus$ denotes the difference.

We define the following operators on sets of roles:

- $\cap$ denotes the intersection,
- $\cup$ denotes the union,
- $\setminus$ denotes the difference.

We define the following operators on sets of users:

- $\cap$ denotes the intersection,
- $\cup$ denotes the union,
- $\setminus$ denotes the difference.

We define the following operators on sets of roles:

- $\cap$ denotes the intersection,
- $\cup$ denotes the union,
- $\setminus$ denotes the difference.

We define the following operators on sets of users:

- $\cap$ denotes the intersection,
- $\cup$ denotes the union,
- $\setminus$ denotes the difference.

We define the following operators on sets of roles:

- $\cap$ denotes the intersection,
- $\cup$ denotes the union,
- $\setminus$ denotes the difference.

We define the following operators on sets of users:

- $\cap$ denotes the intersection,
- $\cup$ denotes the union,
- $\setminus$ denotes the difference.

We define the following operators on sets of roles:

- $\cap$ denotes the intersection,
- $\cup$ denotes the union,
Roles Analysis.

We define the following properties to detect and analyze abnormalities in the set of roles.

- **Property 1 [Redundant Roles]:** two roles \( r_i, r_j \in ROLES_{IMP} \) are redundant if and only if \( PermissionsOfRole(ri) = PermissionsOfRole(rj) \).

We compute in (23) the set of redundant roles. The security architect should remove the redundancy and adjust assignment relations linked to redundant roles.

\[ \text{RedundantRoles} = \{(r_i, r_j) \in ROLES_{IMP} | PermissionsOfRole(r_i) = PermissionsOfRole(r_j) \} \]  

(23)

- **Property 2 [Disjoint Roles]:** two roles \( r_i, r_j \in ROLES_{IMP} \) are disjoint if and only if \( PermissionsOfRole(ri) \cap PermissionsOfRole(rj) = \emptyset \).

We compute in (24) the set of disjoint roles. This gives an overview on the defined hierarchy of roles.

\[ \text{DisjointRoles} = \{(r_i, r_j) \in ROLES_{IMP} | PermissionsOfRole(r_i) \cap PermissionsOfRole(rj) = \emptyset \} \]  

(24)

- **Property 3 [Intersecting Roles]:** two roles \( r_i, r_j \in ROLES_{IMP} \) are intersecting if and only if \( PermissionsOfRole(r_i) \cap PermissionsOfRole(rj) \neq \emptyset \).

We compute in (25) the set of intersecting roles. The security architect should verify the intersecting roles to control redundant assignments and to detect illegal assignments or delegation of privileges.

\[ \text{IntersectingRoles} = \{(r_i, r_j) \in ROLES_{IMP} | PermissionsOfRole(r_i) \cap PermissionsOfRole(rj) \neq \emptyset \} \]  

(25)

- **Property 4 [Absorbing Roles]:** for all roles \( r_i, r_j \in ROLES_{IMP} \), we consider that \( r_i \) is an absorber for \( r_j \) if and only if \( PermissionsOfRole(r_i) \subseteq PermissionsOfRole(rj) \).

We compute in (26) the set of absorbing roles. The security architect should verify the absorbing roles to control the power of the concerned users.

\[ \text{AbsorbingRoles} = \{(r_i, r_j) \in ROLES_{IMP} | PermissionsOfRole(r_i) \subseteq PermissionsOfRole(rj) \} \]  

(26)

### 5. Illustrative Example

We consider the Medical Information System used in [17] as an example. Its functional part defines three components: patients, doctors, and medical records. Each medical record belongs to one patient. Medical records are managed only by doctors responsible for the corresponding patients. The specified security part defines five users: two nurses Ali and Bob, two doctors Dora and Davis, and Paula as a secretary. Doctors and nurses are part of medical staff.

Let suppose for the next that the specification is valid and after a period of time from the implementation, the policy has evolved to a new state where significant changes are introduced. At this step, the security architect proceeds to extract and formalize the concrete policy via applying our reverse engineering and model transformation approach [14].

When checking the equivalence between high and low levels policies, our process detects the anomalies (missed users; hidden users, roles and access flow; redundancy) presented in Figure 6. Based on possible interpretations that emphasize for legal changes, the security architect updates the specification and/or the implementation to reach the equivalence.

![Figure 6. Example of non-compliance cases.](image-url)
6. Conclusions and Future Work

We address in this paper the formal analysis of the integrity of concrete access control policies within relational databases. We mainly focus on the monitoring of the conformity of low level instances of RBAC policies. We define a formal framework that incorporates formal verification, validation and analysis techniques to address the issue. Ongoing works focus on the definition of a formal framework for evaluating the risk associated to the detected anomalies.

References


Faouzi Jaidi received the engineering degree in computer science with Distinction from EABA in 2005. He received his Master's degree in computer science with Distinction from the Faculty of Science, Mathematics, Physics and Natural of Tunis in 2010. He received a PhD degree in ICT with Distinction from the Higher School of Communication of Tunis (Sup’Com), in 2016. Faouzi JAIDI is currently an Assistant Professor at ESPRIT School of Engineering, Tunis, Tunisia. His professional experience from 2005 till now concerns mainly information systems and software security, networks administration and security, formal methods, databases, etc. Actually, he is a member of the Digital Security Research Lab (DSRL) at Sup’Com and a member of MINOS research group at ESPRIT. He is also a member of the Tunisian Society for Digital Security.

Faten Ayachi received in 1987 the Diploma degree in computer management with Distinction from the Higher Institute of Management of Tunis (Tunisia). In 1988, she received a Master’s degree and in 1992 a PhD degree with Distinction from UNSA, University of Nice Sophia Antipolis (France). Faten is currently an Assistant Professor at the Sup’Com Engineering School of Telecommunications in Tunisia. She is a member of the Digital Security Research Lab (DSRL) and member of the Tunisian Society for Digital Security. Her main research areas are Information System Security and databases.

Adel Bouhoula received in 1990 the Diploma degree in computer engineering with Distinction from the University of Tunis (Tunisia). In 1991, he received a Master’s degree, in 1994 a PhD degree with Distinction and in 1998 the Habilitation degree all in computer science from Henri Poincare University in Nancy (France). Adel BOUHOULA is currently a Professor at the Sup’Com Engineering School of Telecommunications in Tunisia and a Visiting Professor at Tsukuba University in Japan. He is the Founder and Head of the Digital Security Research Lab, the Founder and President of the Tunisian Society for Digital Security.