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 Langage Secure (UML) [18] and to verify the obtained models by using the MOdeling, 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.
2. Detecting redundancy and inconsistency anomalies [8] based on graphs of roles.

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): $PERMISSIONS \subseteq ACTIONS \times OBJECTS$.

- AUR, describes in (1) the users-roles assignments: (1) AUR: USERS → ROLES *i.e.,* AUR ⊆ USERS × ROLES
- ARR, describes in (2) the roles-roles assignments: (2) ARR: ROLES → ROLES *i.e.,* ARR ⊆ ROLES × ROLES
- APR, defines in (3) permissions-roles assignments: (3) APR: PERMISSIONS → ROLES *i.e.* APR ⊆ PERMISSIONS × ROLES

We note also ACP_IMP = (USERS_IMP, ROLES_IMP, PERMISSIONS_IMP, AUR_IMP, ARR_IMP, APR_IMP) the formal representation of the concrete policy.

4.1.1. 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.

4.1.2. 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 $r \in ROLES\_{IMP}$ (resp. $r \in ROLES$) the set of permissions assigned to it.

$$PermissionsOfRole(r) = \begin{cases} \{p_i\} \in PERMISSIONS\_{IMP} \mid \\ r \in ROLES\_{IMP} \wedge (r, p_i) \in APR\_{IMP}. \\ \\ \{p_i\} \in PERMISSIONS \mid \\ r \in ROLES \wedge (r, p_i) \in APR. \end{cases} \quad (1)$$

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

$$RolesOfUser(u) = \begin{cases} \{r_i\} \in ROLES\_{IMP} \mid \\ u \in USERS\_{IMP} \wedge (u, r_i) \in AUR\_{IMP}. \\ \\ \{r_i\} \in ROLES \mid \\ u \in USERS \wedge (u, r_i) \in AUR. \end{cases} \quad (2)$$

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

$$PermissionsOfUser(u) = \cup_{r_i \in RolesofUser(u)} PermissionsOfRole(r_i). \quad (3)$$

2. Anomalies Detection.

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

$$HiddenUsers = USERS\_{IMP} - USERS. \quad (4)$$

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

$$HiddenRoles = ROLES\_{IMP} - ROLES. \quad (5)$$

- 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).
  - *HAUR*: logically, this anomaly is detectable when $AUR\_{IMP} - AUR \neq \emptyset$. We compute HAUR in (6) as the difference between the two sets of implemented and specified assignments of users to roles.
  $$HAUR = AUR\_{IMP} - AUR. \quad (6)$$
  - *HARR*: logically, this situation is obvious when $ARR\_{IMP} - ARR \neq \emptyset$. We compute in (7) the set of HARR as the difference between the sets of implemented and specified roles to roles assignments.
  $$HARR = ARR\_{IMP} - ARR. \quad (7)$$
  - *HAPR*: this case is visible when $APR\_{IMP} - APR \neq \emptyset$. We compute in (8) the set of HAPR as the difference between the two sets of implemented and specified permissions to roles assignments.
  $$HAPR = APR\_{IMP} - APR. \quad (8)$$

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.

$$HiddenACFlow = HAUR \cup HARR \cup HAPR. \quad (9)$$

- Detecting Missed Users: missed users are initially specified users but not defined in the concrete policy. This is identified

when $USERS - USERS_{\_IMP} \neq \emptyset$. In (10), we calculate the set of *Missed Users* as the difference between the sets of specified and implemented users.

$$MissedUsers = USERS - USERS_{\_IMP}. \quad (10)$$

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

$$MissedRoles = ROLES - ROLES_{\_IMP}. \quad (11)$$

- Detecting Missed Access Flow: missed access flow concerns Missed Assignments of Roles to Roles (MARR), Roles to Users (MAUR) and Permissions to Roles (MAPR).
  - *MAUR*: logically, this case is detectable when $AUR - AUR_{\_IMP} \neq \emptyset$. In (12), we compute MAUR as the difference between the two sets of specified and implemented assignments of users to roles.

$$MAUR = AUR - AUR_{\_IMP} \quad (12)$$

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

$$MARR = ARR - ARR_{\_IMP} \quad (13)$$

  - *MAPR*: logically, this case is perceptible when $APR - APR_{\_IMP} \neq \emptyset$. We compute MAPR in (14) as the difference between the two sets of specified and implemented assignments of permissions to roles.

$$MAPR = APR - APR_{\_IMP} \quad (14)$$

Similarly, we consider the missed access flow as the union of enumerated sets and we compute it in (15) as the union of the sets of missed assignments.

$$MissedACFlow = MAUR \cup MARR \cup MAPR \quad (15)$$

- Detecting Renamed Users: it belongs to users whose names have been changed but still dispose of the same privileges. We detect this when $\exists\, u_i \in MissedUsers, u_j \in HiddenUsers \mid \forall\, r, ((u_i, r) \in AUR \wedge (u_j, r) \in AUR_{\_IMP})) \vee ((u_i, r) \notin AUR \wedge (u_j, r) \notin AUR_{\_IMP}))$, and we compute it in (16) as the set of couples of hidden and missed users sharing the same permissions and roles.

$$RenamedUsers = \{(u_i, u_j) \mid u_i \in MissedUsers \wedge u_j \in HiddenUsers \wedge \big(PermissionsOfUser(u_i) = PermissionsOfUser(u_j)\big) \wedge \big(RolesOfUser(u_i) = RolesOfUser(u_j)\big)\} \quad (16)$$

- Detecting Renamed Roles: it regroups roles whose names have been changed but still dispose of the same privileges. We identify this when: $\exists\, r_i \in MissedRoles, r_j \in HiddenRoles \mid \forall\, p = (o, a), ((r_i, o, a) \in APR \wedge (r_j, o, a) \in APR_{\_IMP})) \vee ((r_i, o, a) \notin APR \wedge (r_j, o, a) \notin APR_{\_IMP}))$, and we compute it in (17) as the set of couples of hidden and missed roles sharing the same permissions.

$$RenamedRoles = \{(r_i, r_j) \mid r_i \in MissedRoles \wedge r_j \in HiddenRoles \wedge PermissionsOfRole(r_i) = PermissionsOfRole(r_j)\} \quad (17)$$

- Detecting Elementary Redundancy: the elementary redundancy is caused by transitivity and regroups redundant (by transitivity) access rules. This is visible when $\exists u \in USERS_{\_IMP}, r_i, r_j \in ROLES_{\_IMP} \mid (u, r_i) \in AUR_{\_IMP} \wedge (u, r_j) \in AUR_{\_IMP} \wedge (r_i, r_j) \in ARR_{\_IMP}$, and we compute it as the set of triplets expressed in (18).

$$Redundancy = \{(u, r_i, r_j) \mid (u, r_i) \in AUR_{\_IMP} \wedge (u, r_j)\ AUR_{\_IMP} \wedge (r_i, r_j) \in ARR_{\_IMP}\} \quad (18)$$

- Detecting Redundancy Associated to DAC Model: this redundancy is caused by the simultaneous use of RBAC and DAC models. Thus, by using RBAC, we assign permissions to users via roles, while by using DAC we directly assign the same permissions to the same users.

We define in (19) the relation $APU_{\_IMP}$ that illustrates the direct assignment of permissions to users.

$$APU_{\_IMP}: PERMISSIONS_{\_IMP} \rightarrow USERS_{\_IMP} \text{ i.e. } APU_{\_IMP} \subseteq PERMISSIONS_{\_IMP} \times ROLES_{\_IMP} \quad (19)$$

$APU_{\_IMP}$ defines triplets (u, o, a) that represent a direct assignment of the permission (action a on object o) to the user u. Logically, this redundancy is perceptible when $\exists\, u \in USERS_{\_IMP}, r \in ROLES_{\_IMP}, p = (o, a) \mid (u, r) \in AUR_{\_IMP} \wedge (r, o, a) \in APR_{\_IMP} \wedge (u, o, a) \in APU_{\_IMP}$, and we compute it in (20) as the set of the defined quadruplets.

$$DACRedundancy = \{(u, r, o, a) \mid (u, r) \in AUR_{\_IMP} \wedge (r, o, a) \in APR_{\_IMP} \wedge (u, o, a) \in APU_{\_IMP}\} \quad (20)$$

The validation process uses the defined validation properties and formulas to check the conformity of a concrete policy regarding its specification [12].

4.1.3. Formal Analysis

Correctness and completeness proofs.

To prove the correctness of our reasoning, we consider the following cases of conformity analysis.

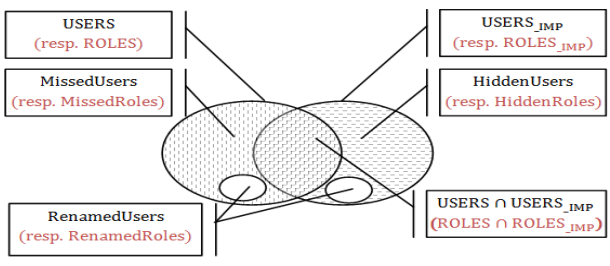


Figure 1. Analysis of the conformity of users (resp. Roles).

(1) $u \vdash$ conformity **iff** $u \in (USERS \cap USERS_{IMP})$.
(2) $u \vdash \neg$conformity **iff** $u \notin (USERS \cap USERS_{IMP})$.
(3) $(USERS, USERS_{IMP}) \models$ conformity **iff** $MissedUsers = \emptyset \wedge HiddenUsers = \emptyset \wedge RenamedUsers = \emptyset$.
(4) $(USERS, USERS_{IMP}) \models \neg$conformity **iff** $MissedUsers \neq \emptyset \vee HiddenUsers \neq \emptyset \vee RenamedUsers \neq \emptyset$.

Figure 2. Inference system: users analysis.

(1) $r \vdash$ conformity **iff** $r \in (ROLES \cap ROLES_{IMP})$.
(2) $r \vdash \neg$conformity **iff** $r \notin (ROLES \cap ROLES_{IMP})$.
(3) $(ROLES, ROLES_{IMP}) \models$ conformity **iff** $MissedRoles = \emptyset \wedge HiddenRoles = \emptyset \wedge RenamedRoles = \emptyset$.
(4) $(ROLES, ROLES_{IMP}) \models \neg$conformity **iff** $MissedRoles \neq \emptyset \vee HiddenRoles \neq \emptyset \vee RenamedRoles \neq \emptyset$.

Figure 3. Inference system: roles analysis.

- Users and Roles Compliance: Figure 1 shows possible cases for the compliance analysis of the sets of users (resp. Roles). The inference systems (figures 2 and 3) allow reasoning on the conformity issues relative to the sets of Users (resp. Roles).

For all users ($\forall u$), we have three possible cases:

1. $u \in USERS \wedge u \notin USERS_{IMP}$,
2. $u \notin USERS \wedge u \in USERS_{IMP}$,
3. $u \in USERS \wedge u \in USERS_{IMP} \equiv u \in (USERS \cap USERS_{IMP})$.

- Conformity proof: we have conformity between the sets of users if $\forall u, u \vdash$ conformity.

If $\forall u, u \vdash$ conformity then by applying the first rule we have $\forall u, u \in (USERS \cap USERS_{IMP})$ which means $USERS = USERS_{IMP} = USERS \cap USERS_{IMP}$. So, $MissedUsers = \emptyset$, $HiddenUsers = \emptyset$ and $RenamedUsers = \emptyset$. Hence, the third rule gives $(USERS, USERS_{IMP}) \models$ conformity.

- Non-Conformity proof: we have non-conformity between the sets of users if $\exists u, u \vdash \neg$conformity.

If $\exists u, u \vdash \neg$conformity then by applying the second rule we have $\exists u, u \notin (USERS \cap USERS_{IMP})$ which means that $\exists u, u \in \{USERS \setminus USERS_{IMP}\} \vee u \in \{USERS_{IMP} \setminus USERS\}$. So, $u \in$ Missed Users $\vee$ $u \in$ HiddenUsers $\vee$ $u \in$ Renamed Users. By consequence we have $MissedUsers \neq \emptyset$ or $HiddenUsers \neq \emptyset$ or $RenamedUsers \neq \emptyset$. According to the forth rule, we have $(USERS, USERS_{IMP}) \models \neg$conformity. Therefore, our reasoning is correct. Idem, for the set of Roles, we easily prove the correctness of our reasoning.

- Assignments Compliance: Figure 4 presents possible cases for compliance analysis of the assignments relations. The inference system in Figure 5 allows reasoning on the conformity issues relative to users-roles assignments (AUR and AUR_{IMP}), while it may be generalized and adapted to all assignment relations.

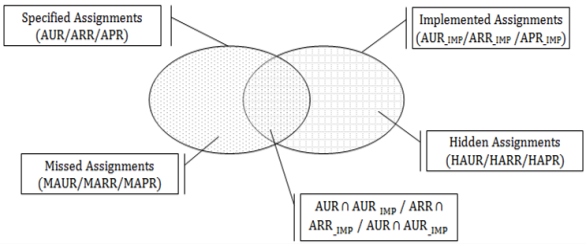


Figure 4. Analysis of the conformity of assignments.

(1) $(u, r) \vdash$ conformity **iff** $(u, r) \in (AUR \cap AUR_{IMP})$.
(2) $(u, r) \vdash \neg$conformity **iff** $(u, r) \notin (AUR \cap AUR_{IMP})$.
(3) $(AUR, AUR_{IMP}) \models$ conformity **iff** $MAUR = \emptyset \wedge HAUR = \emptyset$.
(4) $(AUR, AUR_{IMP}) \models \neg$conformity **iff** $MAUR \neq \emptyset \vee HAUR \neq \emptyset$.

Figure 5. Inference system: users-roles assignments analysis.

For all assignment relations of roles to users ($\forall (u, r)$), we have three possible cases:

1. $(u, r) \in AUR \wedge (u, r) \notin AUR_{IMP}$,
2. $(u, r) \notin AUR \wedge (u, r) \in AUR_{IMP}$,
3. $(u, r) \in AUR \wedge (u, r) \in AUR_{IMP} \equiv (u, r) \in (AUR \cap AUR_{IMP})$.

- Conformity Proof: we have conformity between the sets of assignments if $\forall (u, r), (u, r) \vdash$ conformity.

If $\forall (u, r), (u, r) \vdash$ conformity then by applying the first rule we have $\forall (u, r), (u, r) \in (AUR \cap AUR_{IMP})$ which means that $AUR = AUR_{IMP} = AUR \cap AUR_{IMP}$ and consequently $MAUR = \emptyset$ and $HAUR = \emptyset$. Hence, according to the third rule, we have $(AUR, AUR_{IMP}) \models$ conformity.

- Non-Conformity Proof: we have non-conformity between the sets of assignments if $\exists (u, r), (u, r) \vdash \neg$conformity.

If $\exists (u, r), (u, r) \vdash \neg$conformity then by applying the second rule we have $\exists (u, r), (u, r) \notin (AUR \cap AUR_{IMP})$ which means that $\exists (u, r), (u, r) \in \{AUR \setminus AUR_{IMP}\} \vee (u, r) \in \{AUR_{IMP} \setminus AUR\}$. This implies that $(u, r) \in MAUR \vee (u, r) \in HAUR$ and by consequence we have $MAUR \neq \emptyset$ or $HAUR \neq \emptyset$. Thus, according to the forth rule, we have $(AUR, AUR_{IMP}) \models \neg$conformity. Therefore, our reasoning is correct.

As for the completeness, we note that intuitively a policy based on the $RBAC_1$ model is a collection of finite sets and relations among them. The difference between the initial and final states of that policy is evaluated as the difference between the initial and the final sets/relations. Since the difference between sets is not commutative, the set of defined operators in this

contribution is considered complete with respect to the $RBAC_1$ model.

Analysis of the Expansion of Powers.

It is essential for the security architect and *administrator* to have a dashboard for monitoring and detecting changes in users and roles powers. We refer to functions *(Permissions Of User*(*u*), *Roles Of User*(*u*) and *Permissions Of Roles*(*r*)) to evaluate users and roles power. An important aspect is to have a special view on hidden components and hidden access rules injected in the policy. We introduce the following definitions to compute the hidden power of users and roles.

- Definition 6 [Hidden Power of a role]: we define in (21) the hidden power of a role in terms of hidden authorizations allocated to it.

$$HiddenPower(r) = \{p \in PERMISSIONS\_IMP \mid (r, p) \in HAPR\} \quad (21)$$

- Definition 7 [Hidden Power of a user]: we define in (22) the hidden power of a user either in terms of hidden roles or hidden authorizations assigned to it.

$$\text{HiddenPower(u)} = \begin{cases} \{r\} \in \text{ROLES}_{\text{IMP}} \mid (u, r) \in \text{HAUR}. \\ \{p\} \in \text{PERMISSIONS}_{\text{IMP}} \mid \exists r \in \text{ROLES}_{\text{IMP}} \\ \quad \wedge (r, p) \in \text{HAPR} \wedge (u, r) \in \text{HAUR}. \end{cases} \quad (22)$$

Roles Analysis.

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

- Property 1 [Redundant Roles]: two roles $r_i, r_j \in ROLES\_IMP$ (resp. $r_i, r_j \in ROLES$) are redundant if and only if *Permissions Of Role*(r_i)=*Permissions Of Role*(r_j).

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

$$RedundantRoles = \{(r_i, r_j)\ ROLES\_IMP \mid PermissionsOfRole(r_i) = PermissionsOfRole(r_j)\} \quad (23)$$

- Property 2 [Disjoint Roles]: two roles $r_i, r_j \in \text{ROLES}_{\text{IMP}}$ (resp. $r_i, r_j \in$ ROLES) are disjoint if and only if $\text{PermissionsOfRole}(r_i) \cap \text{PermissionsOfRole}(r_j) = \emptyset$.

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

$$DisjointRoles = \{(r_i, r_j) \in ROLES\_IMP \mid PermissionsOfRole(r_i) \cap PermissionsOfRole(r_j) = \emptyset\} \quad (24)$$

- Property 3 [Intersecting Roles]: two roles $r_i, r_j \in \text{ROLES}_{\text{IMP}}$ (resp. $r_i, r_j \in$ ROLE) are intersecting if and only if $\text{Permissions Of Role}(r_i) \cap \text{Permissions Of Role}(r_j) \neq \emptyset$.

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.

$$IntersectingRoles = \{(r_i, r_j) \in ROLES\_IMP \mid PermissionsOfRole(r_i) \cap PermissionsOfRole(r_j) \neq \emptyset\} \quad (25)$$

- Property 4 [Absorbing Roles]: for all roles $r_i, r_j \in \text{ROLES}_{\text{IMP}}$ (resp.$r_i, r_j \in$ ROLES), we consider that r_i is absorbent for r_j if and only if $\text{Permissions Of Role}(r_j) \subset \text{Permissions Of Role}(r_i)$.

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.

$$AbsorbingRoles = \{(r_i, r_j) \in ROLES\_IMP \mid PermissionsOfRole(r_j) \subset PermissionsOfRole(r_i)\} \quad (26)$$

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.

- **HiddenUsers** = {Martin, Marie}.
- **MissedUsers** = {Bob}.
- **Renamed Users** =∅.
- **HiddenRoles** = {MedicalStudent}.
- **MissedRoles** =∅.
- **RenamedRoles** =∅.
- **HiddenACFlow:**
  - HARR = {(Secretary|-> MedicalStaff)}.
  - HAUR = {(Martin|-> {MedicalStudent}), (Paula|-> {Nurse}), (Marie |-> {Secretary})}.
  - HAPR = {(MedicalStudent -> (MedicalRecord |->{modify}))}.
- **MissedACFlow :**
  - MARR =∅.
  - MAUR = {(Bob |-> {Nurse}}.
  - MAPR =∅.
- **Redundancy** =∅.
- **DacRedundancy** = {(Paula, Nurse, (MedicalRecord |->{read}))}.

Figure 6. Example of non-compliance cases.

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

[1] Abrial J., *The B-Book: Assigning Programs to Meanings*, Press Syndicate of the University of Cambridge, 1996.

[2] Ahmed A. and Arputharaj K., "XML Access Control: Mapping XACML Policies to Relational Database Tables," *The International Arab Journal of Information Technology*, vol. 11, no. 6, pp. 532-539, 2014.

[3] Baldwin R., "Naming and Grouping Privileges to Simplify Security Management in Large Databases," *in Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, pp. 116-132, 1990.

[4] Basin D., Clavel M., Doser J., and Egea M., "Automated Analysis of Security-Design Models," *Information and Software Technology*, vol. 51, no. 5, 815-831, 2009.

[5] Bertino E., Ghinita G., and Kamra A., *Access Control for Databases: Concepts and Systems*, Foundations and Trends, 2011.

[6] Ghadi A., Modèle hiérarchique de contrôle d'accès d'UNIX basé sur un graphe de roles, PhD Theses, 2010.

[7] Hansen F. and Oleshchuk V., "Conformance Checking of RBAC Policy and Its Implementation," *in Proceedings of Information Security Practice and Experience Conference*, Singapore, pp. 144-155, 2005.

[8] Huang C., Sun J., Wang X., and Si Y., "Security Policy Management for Systems Employing Role Based Access Control Model," *Information Technology Journal*, vol. 8, pp. 726-734, 2009.

[9] Idani A., Ledru Y., Richier J., Labiadh M., Qamar N., Gervais F., Laleau R., Milhau J., and Frappier M., "Principles of the coupling between UML and formal notations," PhD Thesis, 2011.

[10] Jaïdi F. and Ayachi F., "An Approach to Formally Validate and Verify the Compliance of Low Level Access Control Policies," *in Proceedings of 13th International Symposium on Pervasive Systems, Algorithms, and Networks*, Chengdu, pp. 1550-1557, 2014.

[11] Jaïdi F. and Ayachi F., "The Problem of Integrity in RBAC-Based Policies within Relational Databases: Synthesis and Problem Study," *in Proceedings of the ACM IMCOM 9th International Conference on Ubiquitous Information Management and Communication*, Bali, pp. 1-8, 2015.

[12] Jaïdi F. and Ayachi F., "To Summarize the Problem of Non-Conformity in Concrete RBAC-Based Policies: Synthesis, System Proposal and Future Directives," *NNGT International Journal of Information Security*, vol. 2, pp. 1-12, 2015.

[13] Jaïdi F. and Ayachi F., *International Conference on Computational Intelligence in Security for Information Systems*, Springer International Publishing Switzerland, 2015.

[14] Jaïdi F. and Ayachi F., "A Reverse Engineering and Model Transformation Approach for RBAC-Administered Databases," *in Proceedings of 13th International Conference on High Performance Computing and Simulation*, Amsterdam, pp. 115-122, 2015.

[15] Koch M., Mancini L., and Parisi-Presicce F., "A Graph-Based Formalism for RBAC," *ACM Transactions on Information and System Security*, vol. 5, no. 3, pp. 332-335, 2002.

[16] Lampson B., "Protection," *ACM SIGOPS Operating Systems Review*, vol. 8, no. 1, pp. 18-24, 1974.

[17] Ledru Y., Idani A., Milhau J., Qamar N., Laleau R., Richier J., and Labiadh M., "Taking into Account Functional Models in the Validation of IS Security Policies," *in Proceedings of Advanced Information Systems Engineering Workshops*, London, pp. 592-606, 2011.

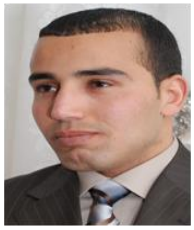
[18] Lodderstedt T., Basin D., and Doser J., "SecureUML: A UML-based Modeling Language for Model-Driven Security," *in Proceedings of 5th International Conference on the Unified Modeling Language*, San Francisco, pp. 426-441, 2002.

[19] Nyanchama M. and Osborn S., "The Role Graph Model and Conflict of Interest," *ACM Transactions on Information and System Security*, vol. 1, no. 2, pp. 3-33, 1999.

[20] Rozenberg G., *Handbook of Graph Grammars and Computing by Graph Transformations*, World Scientific, 1997.

[21] Sandhu R., Coynek E., Feinsteink H., and Youmank C., "Role-Based Access Control Models'," *IEEE Computer*, vol. 29, no. 2, pp. 38-47, 1996.

[22] Thion R. and Coulondre S., "A Relational Database Integrity Framework for Access Control Policies'," *Journal of Intelligent Information Systems*, vol. 38, no.1, pp. 131-159, 2012.



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.