Hindawi
Security and Communication Networks
Volume 2018, Article ID 7142170, 22 pages
https://doi.org/10.1155/2018/7142170

Research Article

A Methodology and Toolkit for Deploying Reliable Security Policies in Critical Infrastructures

Faouzi Jaïdi, Faten Labbene Ayachi, and Adel Bouhoula

Digital Security Research Unit (DSRU), Higher School of Communication of Tunis (Sup'Com), University of Carthage, Tunis, Tunisia

Correspondence should be addressed to Faouzi Jaïdi; faouzi.jaidi@gmail.com

Received 21 October 2017; Revised 22 January 2018; Accepted 13 March 2018; Published 16 May 2018

Academic Editor: Ilaria Matteucci

Substantial advances in Information and Communication Technologies (ICT) bring out novel concepts, solutions, trends, and challenges to integrate intelligent and autonomous systems in critical infrastructures. A new generation of ICT environments (such as smart cities, Internet of Things, *edge-fog-social-cloud* computing, and big data analytics) is emerging; it has different applications to critical domains (such as transportation, communication, finance, commerce, and healthcare) and different interconnections via multiple layers of public and private networks, forming a grid of critical cyberphysical infrastructures. Protecting sensitive and private data and services in critical infrastructures is, at the same time, a main objective and a great challenge for deploying secure systems. It essentially requires setting up trusted security policies. Unfortunately, security solutions should remain compliant and regularly updated to follow and track the evolution of security threats. To address this issue, we propose an advanced methodology for deploying and monitoring the compliance of trusted access control policies. Our proposal extends the traditional life cycle of access control policies with pertinent activities. It integrates formal and semiformal techniques allowing the specification, the verification, the implementation, the reverse-engineering, the validation, the risk assessment, and the optimization of access control policies. To automate and facilitate the practice of our methodology, we introduce our system *SVIRVRO* that allows managing the extended life cycle of access control policies. We refer to an illustrative example to highlight the relevance of our contributions.

1. Introduction

Nowadays, we are witnessing the emergence of new ICT concepts and solutions with the goal of improving the quality of life of citizens and societies. Advances in ICT are at the origin of the spectacular expansion in the development of new cyberphysical systems and services in various fields such as ubiquitous healthcare, smart electricity grid, smart cities, medical monitoring, and process control systems. This new generation of systems is highly interconnected via multiple layers of public and private networks, forming a grid of crucial cyberphysical infrastructures. As an immediate consequence, failure of a component of the grid may result in cascading effect on others. Since they are applied to various critical domains, they are paving the way for new security, reliability, and privacy challenges that cannot be easily solved using traditional techniques but require advanced solutions to be effectively solved.

Protecting private resources against security threats has become a major concern in the development of Information Systems (IS). The access control is well adapted as a strong driving force for protecting data and preserving the privacy. Its importance has been widely studied in literature leading to the definition of three reference models: discretionary (DAC) [1], mandatory (MAC) [2] and role (RBAC) [3] based access control. The great success and the large deployment of the standard model (NIST RBAC) [4] have initiated popular research topics.

Three main properties (security, reliability, and privacy) should be addressed and integrated into the early phases of the implementation of critical cyberphysical infrastructures. Addressing these potential concerns before the infrastructure is in place is a main requirement that may determine the degree of success of a system. As a part of this research theme, relevant works have recommended that developing secure systems must incorporate the security engineering in system and software development life cycles. More deeply, it requires setting up trusted access control policies. Nonetheless, specifying, validating, and implementing a reliable policy and monitoring its compliance have emerged as complex and

confusing tasks. During its life cycle, the policy is highly exposed to inner threats and collusion attempts relative to illegal updates (such as intrusion attempts and delegations of right) or access paradoxically made by authorized users that generate noncompliant changes and incoherent states. A particular crucial problem of inner threats (recognized as security holes which can be exploited by legal users [5]) is related to a malicious use of administrative roles. If administrative roles are not used wisely, a malicious administrator can corrupt the policy and create other security breaches [6]. Moreover, in particular applications such as ubiquitous healthcare and e-healthcare systems, the access control process has to be (at the same time) rigorous to ensure a high level of protection and flexible to treat emergency situations. We verify that coupling two requisite but contradictory objectives (robustness and flexibility) has a direct influence and a wide impact on the compliance of the access control infrastructure [7]. In the context of private databases (used to structure and store collected sensitive and private data about critical and classified environments in cyberphysical infrastructures), typical scenarios of noncompliance between a concrete policy and its specification [8] are defined as follows:

(a) *Hidden users* and *hidden roles* created and granted access rights by an administrator abusing his power.

(b) *Hidden access flow*: users granted the privilege *create any role* or granted roles *with admin option* privilege may delegate those roles to other users. Therefore, they generate a new potential access flow invisible from outside the database.

(c) *Missed users* and *missed roles* due to incorrect use of already granted permissions via role misuse/abuse or due to partial implementation of the policy.

(d) *Missed access flow* due to role misuse/abuse or partial implementation of the policy that falsifies the global behavior of the access control process and may lead the system to deadlock states.

(e) *Renamed users* and *renamed* roles due to malicious administration acts to avoid system audits or investigations.

(f) Redundant, inconsistent, and contradictory access control rules related to malicious acts or to paradoxical policy updates.

To address this problematic, we propose an advanced comprehensive technique for deploying and monitoring the integrity of the implementation of RBAC policies and we focus on its application in the context of private databases (generally managed by relational DataBase Management Systems (DBMS)) as an example. Our methodology enhances the deployment and the management of the conformity of trusted policies. We opted for a formal reasoning to ensure a high level of surety and we reached the following results: (i) the identification and the classification of the anomalies that can alter the policy; (ii) the definition of a formal framework for detecting those anomalies; and (iii) the evaluation of the risk associated with the detected anomalies. A concrete result of our contribution is the implementation of a system that allows automating the practice of our methodology and diagnosing policies defects.

The remainder of the paper is structured as follows. In Section 2, we review and discuss related works. In Section 3, we introduce our methodology for deploying and monitoring the compliance of trusted access control policies. In Section 4, we illustrate the relevance of our contribution based on an example. Finally, in Section 5, we conclude the paper and present ongoing works.

2. Related Works

2.1. RBAC and Security Modeling Languages. The RBAC model [3, 9] introduces the concept of roles (assigned to users) to manage access to resources. This kind of access control based on roles specifies diverse dimensions of RBAC models [3]. The base model (called also the core model) is denoted as $RBAC_0$ and defines the minimum and necessary requirements for a system to support role based access control. The hierarchical RBAC, denoted as $RBAC_1$, introduces the role-hierarchies construct to $RBAC_0$. The constrained model, denoted as $RBAC_2$, extends the core model with the concept of constraints. The consolidated model, tagged as $RBAC_3$, encloses the two concepts of hierarchy and constraint.

As for security modeling languages, SecureUML [10, 11] introduces a new vocabulary to annotate UML class diagrams with access control information. Its methodology relies on the RBAC model extended via authorization constraints expressed in Object Constraint Language (OCL). UMLsec [12, 13] as a security modeling language corresponds to an UML profile that supports various security properties and requirements (such as confidentiality, integrity, and secrecy) illustrated in various UML diagrams by using the concepts of stereotypes, tags, and constraints. Abuse case diagram [14] is another alternative to represent interactions between actors that may result in harmful impact on the system or its actors. Misuse case diagrams [15] allows extending UML traditional use case diagrams via introducing the so-called negative use cases that the system/entity owner does not want to occur. Mal-activity diagram [16] is another attempt that allows representing the normal or the intended behavior of the system and extends it by appending the unintended behavior in order to find remedies for unwanted scenarios.

In conclusion, we can say that SecureUML and UMLsec are both useful to specify and model RBAC policies. As depicted in [17, 18], SecureUML seems to be more suitable for specifying RBAC policies since it has strong features compared to UMLsec. Nevertheless, SecureUML is generally used to model static characteristics of RBAC (applied in the class diagrams) while UMLsec is used to model its dynamic characteristics (it relies heavily on activity diagrams) which means that both approaches complement each other. This complementarity motivated authors in [19] to work on the definition of a model transformation between both languages (SecureUML and UMLsec) for RBAC modeling.

2.2. Verification and Validation of Access Control Policies. Verifying and validating security policies are complex and

difficult tasks. To address this issue, several techniques are proposed in literature, to verify access control policies. As main and objective solutions we highlight proposals based on model checking, algebraic techniques, graph of roles, Satisfiability Modulo Theories (SMT) and Satisfiability (SAT) solver, formal methods, abductive reasoning, and the conjunctions of those techniques. Authors in [20] present synthesis, discussion, and analysis of several popular verification techniques.

Various works treated the verification of access control policies during the specification phase of a system to check the exactitude of the specification before proceeding to its implementation. Authors in [21] proposed to validate the specification (SecureUML diagrams) via OCL requests by using the SecureMOVA tool. In [22], authors formalize the specified diagrams (SecureUML diagrams) in the Z language and analyze the policy based on the specification animator Jaza tool. Authors in [23] suggested translating the SecureUML diagrams in the B notation by using the B4Msecure tool and checking the obtained models via the ProB tool. In [24], authors structure the set of roles in a graph that captures different variants of RBAC models. This representation can benefit from well-established results in graph transformations systems [25] and from issues addressed in [26, 27]. Other verification techniques used model checking for policy validation. Proposals are based on model checking [28–30] and parametrized model checking [31]. Other authors proposed to combine model checking with other techniques such as SMT solvers [32, 33]. Researchers in [34, 35] used algebraic techniques as verification approaches that capture authorizations and obligations and allow identifying anomalies. SMT and SAT solvers are commonly used in policies analysis. Authors in [36, 37] present analysis techniques based on SMT solvers for analyzing XACML and variant of RBAC policies. Abductive reasoning solutions were also proposed to reason out policies conflicts and gaps. This technique is used alone like in [38] to explain access grants, denials, automated delegation, and changes or combined with other techniques such as free variable tableaux [39] to analyze and detect policies conflict for access control in web services environments.

As regards the verification of concrete policies, several research studies fit into this topic with a main goal of verifying the correctness of the implementation regarding the set of predefined constraints. The basic contributions deal with (i) checking the validity of the implemented policy in comparison to its security constraints [40] by using a finite model checker; (ii) detecting anomalies of redundancy and inconsistency in the expression of the policy [41] by utilizing the concept of graph of roles; and (iii) proposing [42] a logical framework used to set and check policy constraints in the case of relational databases. To detect possible intrusions that may threaten an access control system, the author in [43] proposed to model the policy as a graph of roles and to use algorithms of graph theory or LDAP directory schema, to detect illicit transfer of privileges.

2.3. Reverse-Engineering and Formalization of Access Control Policies

Formal Specification of Access Control Policies. The formalization of the functional model corresponds to a classic transformation from UML to the B notation that was addressed by several works such as [44–47]. To take advantage of various complementary research works, authors in [22] proposed to incorporate their translation rules in a unified framework. Concerning the formalization of the security model, it relies on a SecureUML-B [46] mapping which leads to a structure that represents data types. To carry out this step, we use (after defining the necessary adjustments) the B4Msecure tool [48] for transforming UML class diagrams to B formalizations.

Reverse-Engineering of Concrete Policies. Divers works addressed the reverse-engineering in the context of databases that led to the development of professional tools. Authors in [49] present a comparison between most recognized tools. Existing tools allow generating the functional model of an implemented system (database), while they do not allow generating the complete security model. In other words, they do not provide the opportunity to extract all the components of a persistent RBAC policy. Recently, authors in [50] worked on the reverse-engineering of access control policies. This work does not provide too much technical details about the reverse-engineering process and it does not cover all the aspects of RBAC policies such as constraints regeneration.

Encoding a concrete policy (the extracted policy) in a logic-like notation is a preliminary step that allows formally checking the policy concepts and properties. Unfortunately, this thematic is not greatly covered in literature. In our formalization context, researchers addressed the transformation between SQL and B notation from one way that led to the definition of the so-called B-SQL mapping. This mapping is specified in the context of a model driven architecture in order to automate the generation of codes from the specifications. The authors in [51, 52] defined an UML-B-SQL transformation that allows specifying IS with UML, translating the specifications to the formal B notation, and generating after successive refinements Java/SQL codes. The authors in [23] proposed an approach to encode UML and SecureUML diagrams in the formal B notation. According to our knowledge, the transformation in the other way (from SQL to B notation so-called SQL-B mapping) is not addressed in literature.

2.4. Risk Assessment of the Evolutions of Concrete Policies.

Integrating risk awareness in RBAC systems deals mainly with three basic concepts and approaches: (i) enhancing trustworthiness relationships in RBAC systems [53–55]; (ii) defining and enforcing mitigation strategies based on constraints called constraints-based risk mitigation approach; divers attempts have been initiated to identify and formally specify SSOD and DSOD policy constraints [56–59] in RBAC systems; another attempt [60] proposed a mitigation strategy based on risk thresholds and associated obligation pairs; (iii) managing access based on quantified risk values. Many authors discussed and proposed different frameworks in order to quantify risks associated with access control [61–65].

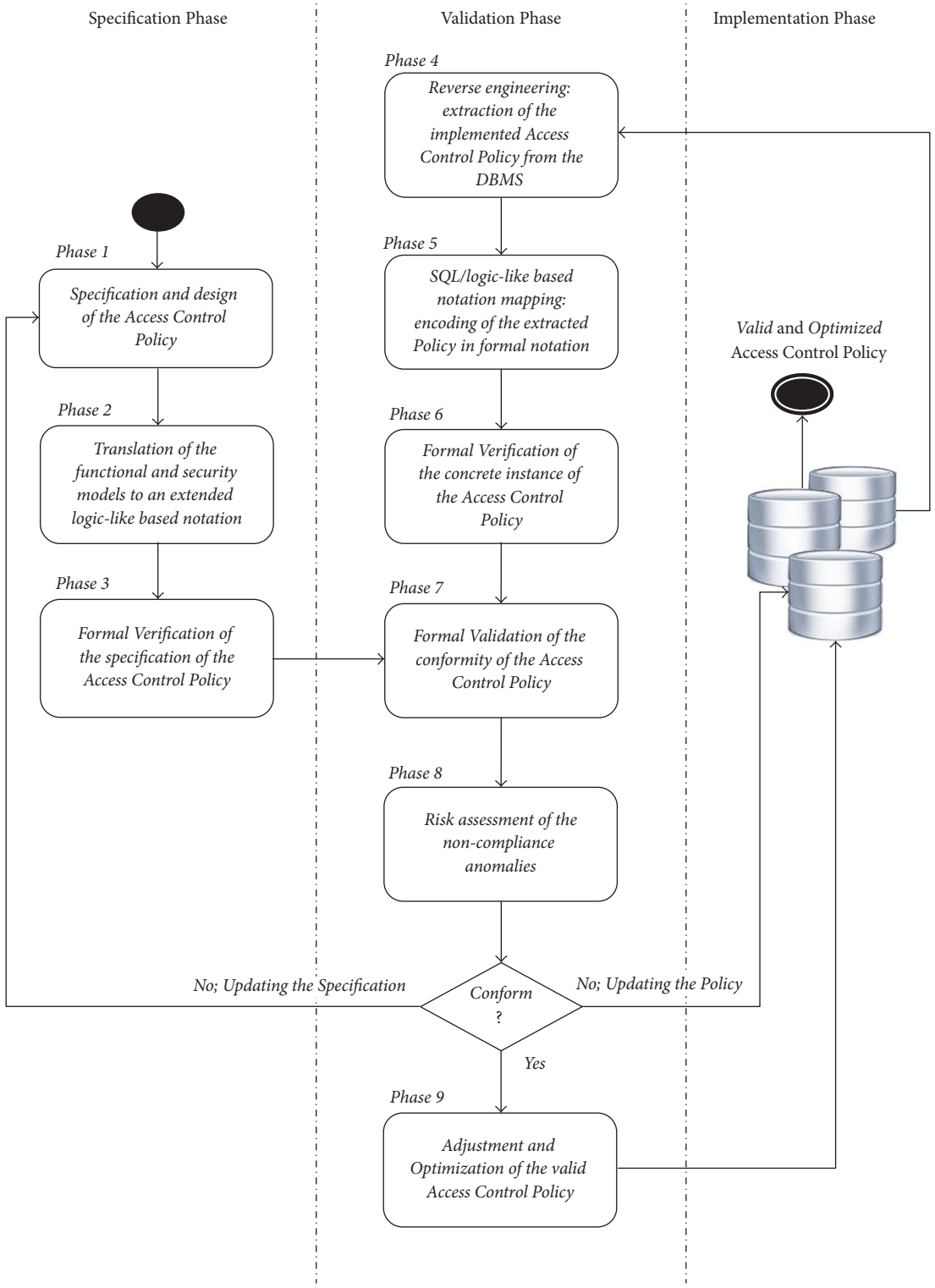


Figure 1: Methodology of deployment and management of the compliance of trusted policies.

3. Deploying and Monitoring the Compliance of Trusted Access Control Policies

3.1. Methodology. As illustrated by Figure 1, our methodology for deploying and monitoring the compliance of a trusted access control policy consists of nine basic phases that define the extended life cycle of security policies.

Phase 1 (specification and modeling of the policy). This first step consists of analyzing and expressing the security

requirements of the system during its specification and design phases. The inputs of this phase are functional and security requirements and its outputs are the specification diagrams (UML diagrams related to both functional and security models). In practice and in our case, we rely on SecureUML [10] as a security modeling language, but it can be based on other security modeling languages.

Phase 2 (formalization of the functional and security models). In this phase, we encode the functional and security models (obtained during the previous phase) in logic-like notations. The strength of a logic-like notation is that it allows a formal verification of the specified policy before proceeding to its concrete implementation and it allows attaining the implementation via applying successive refinements of the specification. This step can be performed by using existing tools such as the B4Msecure tool [48] that allows translating UML diagrams (functional and security models) to B notations [66, 67].

Phase 3 (formal verification of the specified policy). This step consists in formally verifying the exactitude and the coherence of the specification (formalized in the previous phase) before proceeding to its implementation. It can be achieved via different category of tools such as animators and verification tools. In our case, we use the AtelierB and the ProB tools.

Phase 4 (reverse-engineering of the concrete policy). This phase is based on appropriate reverse-engineering techniques that allows extracting the policy enforced by the DBMS. The goal of this phase is to cover the hole mechanisms used for managing access rights. The security requirements relative to the concrete policy are expressed in specific Data Definition Language (DDL) commands. Thus, this phase aims to regenerate the corresponding DDL scripts describing the security model (the access control policy stored in the DBMS dictionary) as well as the functional model.

Phase 5 (formalization of the concrete policy). This phase defines the corresponding mapping SQL/logic-like based notation. Its goal is to encode the generated DDL scripts (obtained during the previous phase) in the target logic-like based notation. In this step, we define the corresponding transformation rules (necessary to achieve this process) based on model-to-model transformation mechanisms such as QVTO (Query View Transformation Operational) technique. The formalization procedure takes as inputs the SQL scripts and generates as outputs the corresponding functional and security B machines (since we utilize the B notation).

Phase 6 (formal verification of the concrete policy). It consists in formally verifying the exactitude and the coherence of the concrete instance of the policy. By analogy to verification of the specified instance of the policy, this step is also based on AtelierB and ProB tools.

Phase 7 (formal validation of the conformity of the policy). At this main step starts the process of validating the compliance between the two instances of the policy: the concrete instance referred to as B implementation machine and its specification referred to as B abstract machine. This defined process has to detect noncompliance situations in case the policy has been evolved to a new state according to new security and functional requirements perhaps for corruption purposes. To do so, we define the requisite validation properties for this process, we analyze and discuss the obtained results, and we propose possible solutions and updates to avoid ambiguities.

Phase 8 (risk assessment of the noncompliance anomalies). This phase allows qualifying (based on quantified risk values) the impact of the detected anomalies on the system in general and particularly on the access control process. It consists in evaluating the risk associated with the detected defects based on a quantified approach. This allows automatic responding to risky anomalies.

Phase 9 (adjustment and optimization of the valid policy). This final step allows correcting anomalies of redundancy and helps *security architects* in checking the properties of the graph of roles, calculating the power of a role, and so on. The security architect has to adjust and update the low level policy based on the obtained results. When this phase is correctly achieved, we should get a valid and optimal instance of the access control policy.

3.2. Specification, Formalization, and Verification of RBAC Policies

3.2.1. Specification of Access Control Policies. We propose in this section an application guide to facilitate the specification of RBAC policies via SecureUML:

(i) *Identifying users*: it consists in determining the set of users and referring to each user or group of users by a class with the stereotype $\ll user \gg$.

(ii) *Identifying application roles*: it consists in fixing the set of roles and referring to each role by a class with the stereotype $\ll role \gg$.

(iii) *Defining roles hierarchy*: it consists in organizing the application roles in a hierarchy that justifies the relationship between roles.

(iv) *Defining constraints between roles*: it consists in fixing the necessary restrictions to manage conflicts between roles such as static and dynamic separation of duties and prerequisite constraints.

(v) *Identifying resources*: it consists in selecting from the functional model the objects (the resources) that require access control rules.

(vi) *Identifying permissions*: it consists in identifying actions that define permissions on resources and referring to permissions via relationships or associations with the stereotype $\ll permission \gg$.

(vii) *Identifying authorization constraints*: it consists in fixing the preconditions that must be met before permissions are granted.

```
(01) ...
(02) SETS
(03)    USERS = {u_1, ..., u_n};
(04)    ROLES = {r_1, ..., r_i};
(05)    OBJECTS = {o_1, ..., o_j};
(06)    ACTIONS = {create, modify, read, delete, ...};
(07) VARIABLES
(08)    UsersRolesAssig,
(09)   RolesHierarchy,
(10)   PermissionsRolesAssig,
(11) ...
(12) INVARIANT
(13)    UsersRolesAssig: USERS − − > POW(ROLES) &
(14)    RolesHierarchy: ROLES < − > ROLES &
       closure1(RolesHierarchy) /\ id(ROLES) = { } &
(15)    PermissionsRolesAssig: ROLES − − > (OBJECTS ∗ POW (ACTIONS)) &
(16) ...
(17) INITIALISATION
(18)    UsersRolesAssig := {(u_1 |− > {r_1}), ..., (u_i |− > {r_j, r_k})} ||
(19)    Roles_Hierarchy := {(r_1 |− > r_k), ..., (r_i |− > r_j)} ||
(20)    PermissionsRolesAssig := {(r_i |− > (o_j |− > {read, modify})), ...} ||
(21) OPERATIONS
(22) ...
(23) END
```

Algorithm 1

(viii) *Defining constraints between permissions*: it consists in fixing necessary restrictions to manage conflicts between permissions such as separation of duties and prerequisite constraints.

(ix) *Assigning permissions to roles*: it consists in attributing a set of permissions to defined role.

(x) *Assigning roles to users*: it consists in associating to each user the corresponding roles that the user may play in the system.

(xi) *Defining cardinality constraints*: it consists in counting and associating cardinality to different relationships defined in the model.

3.2.2. Formalization of Specified RBAC Policies. In literature, different formalizations of RBAC policies are defined in B, Z, and Alloy languages. We adopt the B notation since (i) it is a powerful logic-like language and (ii) it is very common for model driven approaches that allow reaching the implementation after successive refinements of the specification. We denote by *ACP = (USERS, ROLES, PERMISSIONS, AUR, ARR, APR)* the formal notation of the specified RBAC policy, where

(i) *USERS* represents the set of authorized users;

(ii) *ROLES* is the set of predefined roles;

(iii) *OBJECTS* belongs to the set of resources;

(iv) *ACTIONS* corresponds to the set of access modes;

(v) *PERMISSIONS* is the set of permissions (possible actions on objects), denoted as $PERMISSIONS \subseteq ROLES \times OBJECTS$;

(vi) *AUR* represents the set of couples (u_i, r_j) that characterizes the users-roles assignments, denoted as $AUR \subseteq USERS \times ROLES$;

(vii) *ARR* defines the roles-roles assignments as the set of couples (r_i, r_k), denoted as $ARR \subseteq ROLES \times ROLES$;

(viii) *APR* illustrates the assignment of permissions to roles defined by the set of couples (p_i, r_j), denoted as $APR \subseteq ROLES \times OBJECTS \times ACTIONS$.

As previously discussed in the related works section, the formalization of the functional model corresponds to a classic transformation from UML to the B notation that was addressed by several works. As for the formalization of the security model, we based on a SecureUML-B mapping which leads to a structure that represents data types. To carry out this step, we use (after defining the necessary adjustments) the B4Msecure tool for transforming UML class diagrams to B formalizations. This adjustment is necessary due to two main reasons: (i) to preserve the coherence with the formal representation of RBAC policies; (ii) to later minimize the complexity of the verification and validation processes. Therefore, we define essential modifications to the QVTO (Query View Transformation Operational) rule used by the tool in order to get the exact definition of the assignments of permissions to roles. We check that our updates maintain the consistency of the translation process. The encoding of a RBAC policy in the B notation is structured as described by the following example in Algorithm 1.

3.2.3. Formal Verification of RBAC Policies. The formal verification of the specification of an access control policy allows

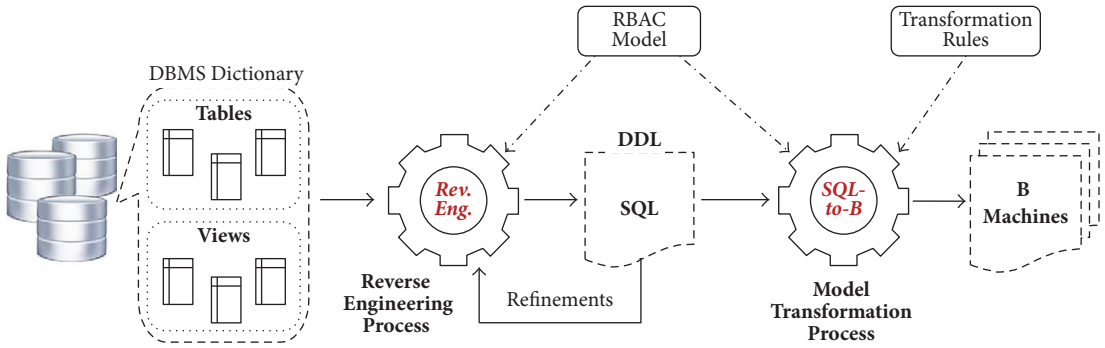


Figure 2: Reverse-engineering and formalization approach of concrete RBAC policies.

```
CREATE USER PAUL IDENTIFIED BY PAUL DEFAULT TABLESPACE
SYSTEM TEMPORARY TABLESPACE TEMP;
```

Box 1

mainly checking the exactitude and the correctness of the specification. To do so, we proceed, in a first step, by verifying that the generated B machines are well structured, coherent, and syntactically and semantically correct, and we check the establishment and the preservation of the invariants. Thus, by using the AtelierB tool, we verify the establishment of the invariants on initialization and during operations calls. This tool allows also type checking, generation, demonstration of proof obligations, and so forth.

Then, in a second step, we proceed by animating the specification via the ProB tool. This animation should report two main characteristics of the system: (i) the behavior of the access control process and (ii) the relevance of the specifications regarding both functional and especially security needs.

3.3. Formalization and Verification of Concrete Policies. Our approach to formalize a concrete instance of a RBAC policy, presented in Figure 2, is based on reverse-engineering techniques for extracting the implemented policy from the Oracle DBMS and model-to-model transformation formalisms for representing the extracted policy in the B notation [68]. We focus on Oracle databases since Oracle is a familiar and a widely used DBMS.

3.3.1. Reverse-Engineering of Concrete Policies. To reverse-engineer a persistent RBAC policy, we have to consider two components of the database: persistent data (functional schema) and persistent access control policy (security schema). Several tools defined in literature like Rational Rose, Power AMC designer, and DBVisualizer can be used to extract the functional schema from a concrete database, but they do not allow generating the security schema. Therefore, we define the necessary SQL reverse-engineering scripts that interrogate the DBMS data dictionary in order to extract and regenerate Data Definition Language (DDL) statements relevant to the implemented policy. The reverse-engineering

```
CREATE ROLE DOCTOR;
CREATE ROLE MEDICALSTAFF;
```

Box 2

scripts proceed in two main steps: (i) scanning corresponding tables and views of the DBMS data dictionary to extract pertinent information relative to the structure of the RBAC model; (ii) generating based on successive refinements DDL scripts (encoded in SQL) that structure the schema of the policy.

The reverse-engineering process [68] generates (in a structured format) from a concrete database the sets of users, the set of roles, the set of objects, the hierarchy of roles, the assignments of roles to users, the assignments of permissions to roles, and specific constraints. In the following, we present a summary of the structure and features of the defined process.

Users Generation. To generate the set of users, we refer to the data dictionary view *dba_users*. An example of output is shown in Box 1.

Roles Generation. To generate the set of roles, we basically refer to the *dba_roles* and to restraint the selection we use the views *role_sys_privs* and *role_tab_privs*. We get an example in Box 2.

Roles Hierarchy Generation. We basically consider the view *dba_role_privs* to generate the hierarchy of roles and we use the views *role_sys_privs*, *role_tab_privs*, and *dba_users* to restraint the selection. An example of output is shown in Box 3.

```
GRANT MEDICALSTAFF TO DOCTOR;
GRANT MEDICALSTAFF TO NURSE;
```

Box 3

```
GRANT DOCTOR TO PAUL;
GRANT NURSE TO MARIE;
```

Box 4

Users-Roles Assignments Generation. To generate roles-users attributions, we refer to the DBA view *dba_role_privs* and we filter the selection by using the *dba_roles* view. An example of output is shown in Box 4.

Roles-Permissions Assignments Generation. To generate the assignment of permissions to roles, we refer to the data dictionary table *role_tab_privs* and we restraint the selection via the *role_sys_privs* view. An example of output is shown in Box 5.

Specific Objects Generation. We mean by specific objects the set of procedures and functions defined in the database schema. To generate specific_objects We refer to the data dictionary table *all_source*. An example of output is shown in Box 6.

Generic Constraints Generation. We refer to the views *all_constraints* and *all_tables* to regenerate generic constraints. An example of output is shown in Box 7.

Authorization Constraints Generation. Authorization constraints are defined through the use of triggers. To generate all the defined triggers, we refer to *all_triggers* or *(dba_triggers)* views. Box 8 is an example of output.

3.3.2. Formalization of Concrete RBAC Policies. The translation of the regenerated functional model is a classic transformation from UML class diagrams to the B language. We use the B4Msecure tool to perform this step. Our translation process of the security model is based on a SQL-B mapping which leads to structure that represents data types. We adopt the proposed B formalization of a variant of the SecureUML metamodel [46]. Then, we define the necessary transformation rules to translate the security model elements in that B specification [68]. We present below, a summary of the transformation rules defined by the process.

Users Transformation. A DDL statement used to generate a user is encoded in the B notation by defining the user as a member of the set *USERS* shown in Box 9.

Roles Transformation. A DDL statement used to generate a role is mapped into B notation by adding this role to the set *ROLES* shown in Box 10.

Roles Hierarchy Transformation. A DDL statement that assigns a role to another role is translated to B notation by inserting both of the two roles as a couple of the function *RolesHierarchy* shown in Box 11.

Users-Roles Assignments Transformation. DDL statements that grant roles to users are mapped into B notation via the *UsersRolesAssg* function that relates the set *USERS* to a subset of the set *ROLES* shown in Box 12.

Roles-Permissions Assignments Transformation. Permissions accorded to roles are translated in B notation by using the sets *OBJECTS* and *ACTIONS* and the function *PermissionsRolesAssig* that associates actions to roles shown in Box 13.

3.3.3. Formal Verification of Concrete RBAC Policies. The formal verification of a concrete RBAC policy concerns the check of the exactitude and the correctness of the implemented policy. We proceed by verifying that the obtained B machines are well structured, coherent, and syntactically and semantically correct. By using the AtelierB tool, we verify the establishment of the invariants on the initialization and during operations calls. We proceed also with type checking, as well as generating and demonstrating the proof obligations.

3.4. Formal Validation of the Conformity of RBAC Policies. To check the compliance of a concrete policy, the formal validation process requires putting in duality two different formal notations. To do so, we denote by *ACP = (USERS, ROLES, PERMISSIONS, AUR, ARR, APR)* the formal representation of the specified policy and we denote by $ACP_{\_IMP}$ = ($USERS_{\_IMP}$, $ROLES_{\_IMP}$, $PERMISSION_{\_IMP}$, $AUR_{\_IMP}$, $ARR_{\_IMP}$, $APR_{\_IMP}$) the formal representation of its concrete instance. The validation phase checks the conformity of the concrete policy by formally comparing it to its specification [69]. We present in the following a summary of the properties of compliance validation.

(1) Hidden Users. It detects new users (not initially defined) injected in the concrete instance of the policy.

$$HiddenUsers = USERS_{\_IMP} - USERS. \tag{1}$$

(2) Missed Users. It detects initially specified users which are not defined in the concrete policy.

$$MissedUsers = USERS - USERS_{\_IMP}. \tag{2}$$

(3) Renamed Users. It regroups the set of users that are renamed. This is detectable via the existence of a couple of missed and hidden users that share the same roles and permissions.

$$\begin{aligned} RenamedUsers = \{&(U_i, U_j) \mid U_i \in MissedUsers \wedge U_j \\ &\in HiddenUsers \wedge PermissionsOfUser(U_i) \\ &= PermissionsOfUser(U_j) \wedge RolesOfUser(U_i) \\ &= RolesOfUser(U_j)\}. \end{aligned} \tag{3}$$

```
GRANT SELECT ON MEDICALRECORD TO MEDICALSTAFF;
GRANT EXECUTE ON PROC_VALIDATE TO DOCTOR;
```

Box 5

```
CREATE OR REPLACE PROCEDURE PROCEDURE1 AS
BEGIN
  ...
END PROCEDURE1;
```

Box 6

```
ALTER TABLE MEDICALRECORD ADD (CHECK (VALID = 0 OR VALID = 1));
```

Box 7

```
CREATE OR REPLACE TRIGGER TRIGGER1
BEFORE UPDATE OF DATA ON MEDICALRECORD
BEGIN
  ...
END;
```

Box 8

```
* SQL:
create user User1 identified by ...;
create user User2 identified by ...;

* B:
SETS
  USERS = {User1, User2};
```

Box 9

```
* SQL:
create role Role1;
create role Role2;

* B:
SETS
  ROLES = {Role1, Role2};
```

Box 10

(4) Hidden Roles. It detects new roles, not initially planned, introduced in the concrete instance.

$$HiddenRoles = ROLES_{\_IMP} - ROLES. \quad (4)$$

(5) Missed Roles. It detects roles initially identified but not implemented or removed.

$$MissedRoles = ROLES - ROLES_{\_IMP}. \quad (5)$$

(6) Renamed Roles. It regroups the set of roles that are renamed. This is visible when a couple of missed and hidden roles share the same permissions.

$$RenamedRoles = \{(R_i, R_j) \mid R_i \in MissedRoles \wedge R_j \in HiddenRoles \wedge PermissionsOfRole(R_i) = PermissionsOfRole(R_j)\}. \quad (6)$$

(7) Hidden AUR. It detects illegal assignments of roles to users injected in the concrete instance.

$$HiddenAUR = AUR_{\_IMP} - AUR. \quad (7)$$

(8) Hidden ARR. It detects illegal hierarchy of roles defined in the concrete instance.

$$HiddenARR = ARR_{\_IMP} - ARR. \quad (8)$$

(9) Hidden APR. It detects hidden assignments of permissions to roles injected in the concrete instance.

$$HiddenAPR = APR_{\_IMP} - APR. \quad (9)$$

```
∗ SQL:
grant Role3 to Role1;
grant Role4 to Role2;

∗ B:
VARIABLES
    RolesHierarchy,. . .
INVARIANT
    RolesHierarchy: ROLES < − > ROLES & . . .
INITIALISATION
    RolesHierarchy := {(Role1|− > Role3), (Role2|− >  Role4)} ‖ . . .
```

Box 11

```
∗ SQL:
grant Role1 to User1;
grant Role2 to User2;
grant Role3 to User2;

∗ B:
VARIABLES
    UsersRolesAssg,. . .
INVARIANT
    UsersRolesAssg: USERS − − > POW (ROLES) & . . .
INITIALISATION
    UsersRolesAssg := {(User1|− > {Role1}), (User2|− > {Role2, Role3})}‖ . . .
```

Box 12

```
∗ SQL:
grant update on table2 to Role2;
grant select on table2 to Role2;
grant execute on procedure to Role3;

∗ B:
SETS
    OBJECTS = {table2, procedure};
    ACTIONS= {read, create, modify, delete, fullAccess, readOp, modifyOp};
VARIABLES
    PermissionsRolesAssig,. . .
INVARIANT
    PermissionsRolesAssig: ROLES − − > (OBJECTS ∗ POW (ACTIONS)) & . . .
INITIALISATION
    PermissionsRolesAssig := {(Role2|− > (table2 |− > {update, select})), (Role3|− >(procedure |− >
        {readOp}))‖ . . .
```

Box 13

(10) Hidden Access Flow. It detects hidden access flow perceptible in the case of illegal assignments of roles to roles, roles to users, or permissions to roles. In general cases, the union operator requires the same typing for all the sets to be combined. Nevertheless, we proceed to types checking for all the defined sets in the verification process. In the validation process, we consider that the types checking is not necessary since it is already done and we

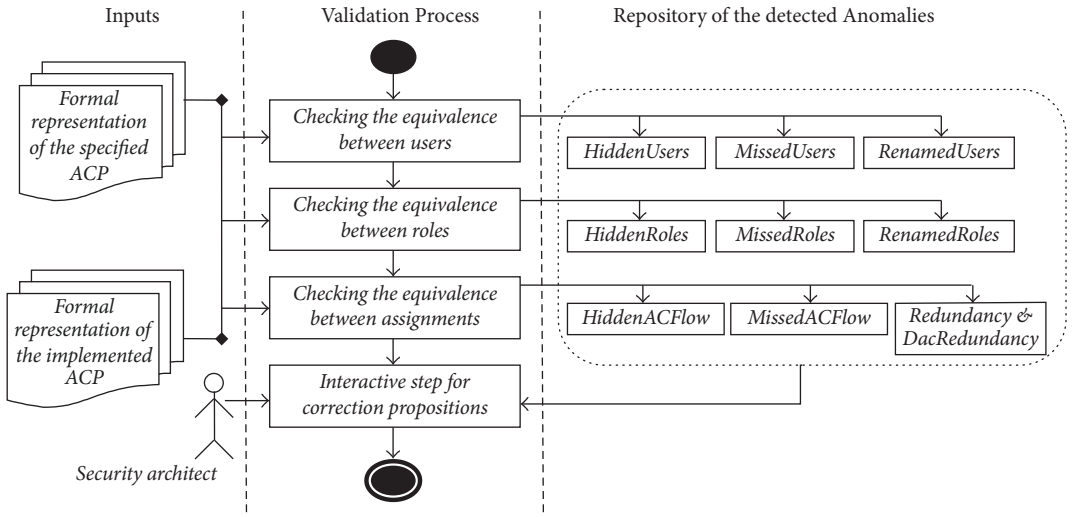


Figure 3: The validation process.

consider the hidden access flow as the union of enumerated sets.

$$HiddenACFlow = HiddenARR \cup HiddenAUR \cup HiddenAPR. \tag{10}$$

(11) Missed AUR. It detects specified assignments of roles to users which are not implemented or removed in the concrete instance.

$$MissedAUR = AUR - AUR_{\_IMP}. \tag{11}$$

(12) Missed ARR. It detects specified hierarchy of roles which are not implemented or removed in the concrete instance.

$$MissedARR = ARR - ARR_{\_IMP}. \tag{12}$$

(13) Missed APR. It detects specified assignments of permissions to roles which are not implemented or removed in the concrete instance.

$$MissedAPR = APR - APR_{\_IMP}. \tag{13}$$

(14) Missed Access Flow. It detects missed access flow perceptible in the case of nonimplementation or removal of assignments of roles to roles, roles to users, or permissions to roles. Idem, in formal notations, the union operator needs the same typing for all the sets to be combined. In the validation process, we do not focus on types checking since it is already done in the verification process and we consider the missed access flow as the union of enumerated sets.

$$MissedACFlow = MissedARR \cup MissedAUR \cup MissedAPR. \tag{14}$$

(15) Generic Redundancy. It expresses redundant access control rules caused by transitivity.

$$Redundancy = \left\{(U_k, R_i) \times \left(U_k, R_j\right) \mid (U_k, R_i) \in AUR_{\_IMP} \wedge \left(U_k, R_j\right) \in AUR_{\_IMP} \wedge \left(R_j, R_i\right) \in ARR_{\_IMP}\right\}. \tag{15}$$

(16) DAC Redundancy. It expresses redundant access control rules related to the assignment of permissions to users via roles using the RBAC model and directly using the DAC model via the relation $APU_{\_IMP}$. This relation is defined as follows:

$$APU_{\_IMP} \subseteq USERS_{\_IMP} \times OBJECTS_{\_IMP} \times ACTIONS_{\_IMP}.$$

$$DacRedundancy = \{(U, R) \times (P, R) \times (U, P) \mid (U, R) \in AUR_{\_IMP} \wedge (P, R) \in APR_{\_IMP} \wedge (U, P) \in APU_{\_IMP}\}. \tag{16}$$

The formal validation process, depicted in Figure 3, checks the equivalence between the concrete policy and its specification [69] based on predefined validation properties. It checks the equivalence between the sets of users, roles, and assignment relations. It stores the detected anomalies in a repository for a real time analysis and for further usage. This process is iterative in the sense that the modifications introduced by the security architect must be checked until no anomaly is detected.

Table 1: The initial risk rating.

Risk rating	Percentage	Description
Extremely high	≥80%	The component evolution is associated with an extremely high risk
High	≥60% and <80%	The component evolution is associated with a high risk
Moderate	≥40% and <60%	The component evolution is associated with a medium risk
Low	≥20% and <40%	The component evolution is associated with a low risk
Minor	≥0% and <20%	The component evolution is associated with a minor risk

3.5. Risk Assessment of the Noncompliance Anomalies. Our risk assessment approach aims to measure the distance of evolution, in terms of risk, between two instances of a security policy [70]. We focused when defining our approach on how to help the security architect to quantify that risk. The risk assessment engine is able to estimate and reestimate a risk threshold or a risk rating for each component based on the predefined risk factors such as history and contextual events. We propose in Table 1 an initial risk rating that will be updated based on the evolution of the risk factors. The choice of five rates is not compulsory and may vary depending on the *security architect* viewpoint.

For each rate we associate a minimum and a maximum *(Rate_MaxPerc*, *Rate_MinPerc)* percentages to limit its borders. The following algorithm allows reestimating the risk rating:

(1) **for all** Rate ∈ RATING **do**
(2) **If** Level(Rate) == MinLevel **then**
(3) Rate_MinPerc ← 0%;
(4) **else**
(5) Rate_MinPerc ← ((Level(Rate) − LevelStep) ∗ 100)/(MaxLevel + (α ∗ CL + β ∗ H + γ ∗ P + ϵ ∗ TR + θ ∗ AR))%;
(6) **end if**
(7) **If** Level(Rate) == MaxLevel **then**
(8) Rate_MaxPerc ← 100%;
(9) **else**
(10) Rate_MaxPerc ← (Level(Rate) ∗ 100)/(MaxLevel + (α ∗ CL + β ∗ H + γ ∗ P + ϵ ∗ TR + θ ∗ AR)) %;
(11) **end if**
(12) **end**

Level (Rate) computes the level of each Rate; *LevelStep* is the step of levels; *MaxLevel* is the highest level; *CL* is the *Criticality Level* of the system; *H* is the *History* risk factor; *P* is the *Purposes* risk factor; *TR* quantifies the probability of risk in an average of time. For instance, access is more risky in the time out of service than in the time of service; *AR* quantifies the probability of risk relative to access types. For instance, access is more risky from outside the office than from the inside; α, β, γ, ϵ, and θ are coefficients that quantify the risk factors.

The risk assessment engine defines a response monitor in order to automatically deactivate risky components. Risky components are identified according to the defined risk thresholds and rating. The monitor classifies the risk associated with each risk-aware component and reacts by deactivating the components based on a threshold fixed by the *security architect*. To automatically deactivate risky hidden and renamed users/roles and revoke risky hidden assignments, the monitor should be able to connect with administrative privileges to the database and execute administrative SQL statements.

We depict in the following a summary of the formal risk assessment formulas used to evaluate the risk values associated with the access control policy components and with the detected noncompliance anomalies.

We evaluate in (17) the risk of a permission $R(P_i)$ as the sum of the probabilities $\Pr(k)$ of occurrence of malicious usages, k, $k = 1, \ldots, m$, multiplied by the cost associated with each malicious usage $C(k)$.

$$R\left(P_i\right) = \sum_{(k=1)}^{m} \Pr\left(k\right) * C\left(k\right). \tag{17}$$

We compute the risk of the role R_j, as illustrated in (18), as the sum of the risk values of all permissions, $R(P_i)$, $i = 0, \ldots, n$, assigned to it.

$$R\left(R_j\right) = \sum_{(i=0)}^{n} R\left(P_i\right) \mid P_i \in APR\left(R_j\right). \tag{18}$$

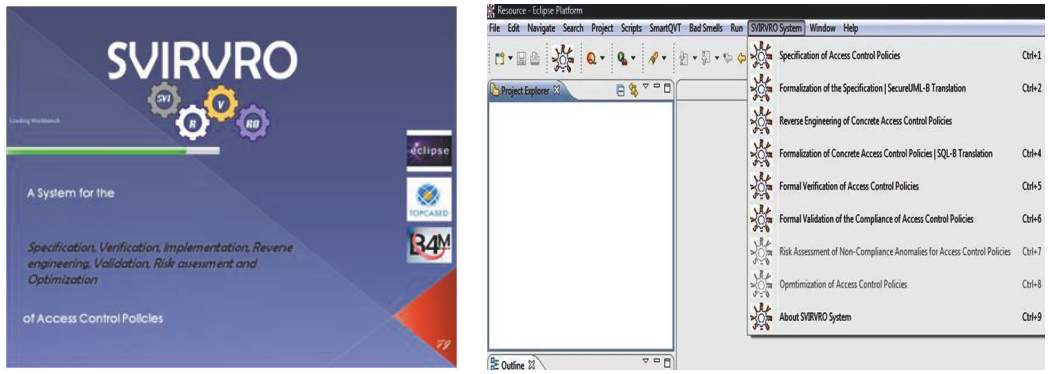
We evaluate the risk of the user $R(U_i)$ as shown in (19) as the sum of the risk values of all roles, R_j, $j = 0, \ldots, n$, assigned to it.

$$R\left(U_i\right) = \sum_{(j=0)}^{n} R\left(R_j\right) \mid R_j \in AUR\left(U_i\right). \tag{19}$$

We consider the risk of an association as the ratio between the risk values of the members of the association. For example, the risk value of the user-role assignment relation $AUR(k)$ that attributes the role R_j to the user U_i is evaluated, as defined in (20), as the ratio between the risk of the role and the risk of the user.

$$R\left(AUR\left(k\right)\right) = \frac{R\left(R_j\right)}{R\left(U_i\right)}. \tag{20}$$

As for the risk assessment of the policy defects, we seek to determine the impact of each anomaly on the system; that is, we probe to quantify the influence and the effect of the associated security breaches on the system. From this perspective, we evaluate the risk of an anomaly, as presented in (21), as the ratio between the risk values of the elements of



(a) Loading (b) Menu

Figure 4: Loading and menu interfaces of SVIRVRO system.

```
Inputs: RBAC-model-structure, Dictionary_views_&_tables.
Outputs: ACP.sql (SQL Script).
Begin
generating_SQL_Script_Format ();
read (RBAC-model-structure);
for each RBAC_Component do
   find_corresponding_views (Dictionary_views_&_tables);
   extract_information_component (RBAC_Component);
   generate_SQL_statements();
end for;
refining_description ();
save_&_return_SQL_Script (ACP.sql);
end
```

Algorithm 2: Reverse engineering access control policies.

this anomaly and the risk values of the system elements of the same type.

$$R\left(Anomaly\right) = \frac{\sum R\left(x\right) \mid x \in Anomaly}{\sum R\left(y\right) \mid y \in System} * 100\%. \quad (21)$$

For example, the risk of the set hidden roles is evaluated in (22) as the sum of the risk values of all hidden roles, R_j, $j = 0, \ldots, n$, divided by the sum of the risk values of all maintained roles. Maintained roles are defined as the intersection between specified and implemented roles.

$$R\left(RenamedRoles\right) = \frac{\sum_{(j=0)}^{n} R\left(R_j\right) \mid R_j \in RenamedRoles}{\sum_{(l=0)}^{m} R\left(R_l\right) \mid R_l \in \left(ROLES_{\_IMP} \cap ROLES\right)} * 100\%. \quad (22)$$

3.6. The SVIRVRO System

3.6.1. Presentation. In order to facilitate the practice of the different phases of our defined methodology, we briefly introduce in this section our system called SVIRVRO *(Specification, Verification, Implementation, Reverse-engineering, Validation, Risk assessment, and Optimization)* for deploying and monitoring the compliance and managing the life cycle of trusted access control policies. This system couples mainly formal and semiformal techniques allowing the management of access control policies life cycle. SVIRVRO is mainly developed as an open source project under the eclipse platform. Figure 4 presents the loading and menu interfaces of the defined system.

Since the reverse-engineering, the formalization of extracted policies, the validation, and the risk assessment phases represent main and major contributions of the system, we briefly cast a glance on the main algorithms relative to the reverse-engineering, the formalization, and the validation processes.

The reverse-engineering algorithm (Algorithm 2) allows generating SQL scripts describing the actual state of the concrete instance of the policy. It proceeds first by scanning the data dictionary views and tables to extract pertinent information relative to each RBAC component. Then, it generates the corresponding SQL statements. Finally, it refines the

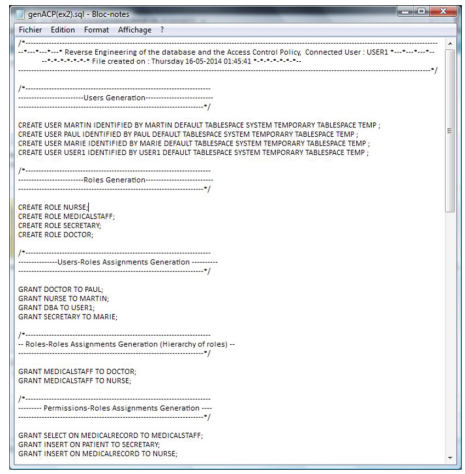


Figure 5: Example of extracted access control policy.

```
Inputs: ACP.sql (SQL Script), Transformation rules.
Outputs: B Machines.
Begin
generating_B_machines_Format ();
read (ACP.sql);
for each SQL statement do
   find_valide_transformation_rule ();
   apply_transformation ();
end for;
save_&_return_B-machines ();
end
```

Algorithm 3: Formalization of extracted access control policies.

description of the extracted policy to preserve the coherence of the generated scripts.

Figure 5 illustrates an example of a generated SQL script relative to an extracted access control policy.

The formalization algorithm (Algorithm 3) aims to translate SQL statements to the corresponding B statements based on the defined transformation rules. It generates B machines that formally represent the concrete instance of the access control policy.

The validation phase, as described in Algorithm 4, checks the equivalence between the sets of users, roles, and assignments relative to both instances of the access control policy. Then it verifies the existence of possible cases of redundancy.

The system generates a report relative to the validation process that notifies all cases of detected anomalies of noncompliance. An example of generated reports (generated in the French language) is depicted by Figure 6.

3.6.2. Evaluation. We address in this section the evaluation and comparison of our toolkit with existing solutions. A main force of our proposal is that it combines a lot of features. It allows the specification, the formalization, the verification, the reverse-engineering, the validation, and the risk assessment. As for the specification of access control policies, our tool is based on a justified approach which is very adequate to the problem statement. Concerning the formalization, we build our process upon a justified proposal and defined necessary refinement to gain more performance and correspondence with the defined objectives. As for the reverse-engineering, our tool is one of the most complete tools that allows regeneration of both functional and security schemes. As for the verification and validation, the force of the tool is that it is based on formal methods for a better reasoning since we formally proved the correctness and the completeness of the proposal. Finally, concerning the risk assessment, our proposal joins novel approaches based on quantified risk assessment. In future works, we will address the performance evaluation of the solution particularly in the context of big and scalable infrastructure.

4. Case Study

In order to illustrate our proposal, we consider a smart city *meeting scheduler*, described in [22], as an illustrative

```
Inputs:{specifications.mch},{implementations.mch}.
Outputs: {HiddenUsers, HiddenRoles,HiddenAUR, HiddenARR, HiddenAPR, HiddenACFlow,
MissedUsers, MissedRoles, MissedAUR, MissedARR, MissedAPR, MissedACFlow, RenamedUsers,
RenamedRoles, Redundancy, DacRedundancy}, conformity.
Begin
read_specifications_machines ();
read_implementations_machines ();
conformity = true;
// checking the equivalence between users
HiddenUsers = calculate_hiddenusers (Users, Users_imp);
MissedUsers = calculate_missedusers (Users, Users_imp);
RenamedUsers = calculate_renamedusers (HiddenUsers, MissedUsers, AUR, AUR_imp, APR, APR_imp);
if (HiddenUsers ≠ ∅ or MissedUsers ≠ ∅ or RenamedUsers ≠ ∅) then
   conformity = false;
end if;
// checking the equivalence between roles
HiddenRoles = calculate_hiddenroles (Roles, Roles_imp);
MissedRoles = calculate_missedroles (Roles, Roles_imp);
RenamedRoles = calculate_renamedroles (HiddenRoles, MissedRoles, APR, APR_imp);
if (HiddenRoles ≠ ∅ or MissedRoles ≠ ∅ or RenamedRoles ≠ ∅) then
   conformity = false;
end if;
// checking the equivalence between users-roles assignments
HiddenAUR = calculate_hiddenAUR (AUR, AUR_imp);
MissedAUR = calculate_missedAUR (AUR, AUR_imp);
if (HiddenAUR ≠ ∅ or MissedAUR ≠ ∅) then
   conformity = false;
end if;
// checking the equivalence between hierarchies of roles
HiddenARR = calculate_hiddenARR (ARR, ARR_imp);
MissedARR = calculate_missedARR (ARR, ARR_imp);
if (HiddenARR ≠ ∅ or MissedARR ≠ ∅) then
   conformity = false;
end if;
// checking the equivalence between permissions-roles assignments
HiddenAPR = calculate_hiddenAPR (APR, APR_imp);
MissedAPR = calculate_missedAPR (APR, APR_imp);
if (HiddenAPR ≠ ∅ or MissedAPR ≠ ∅) then
   conformity = false;
end if;
// checking the equivalence between access flows
HiddenACFlow = calculate_hiddenACFlow (HiddenAUR, HiddenARR, HiddenAPR);
MissedACFlow = calculate_missedACFlow (HiddenAUR, HiddenARR, HiddenAPR);
// checking redundancies
Redundancy = verifyRED();
DacRedundancy = verifyDACRED();
if (Redundancy ≠ ∅ or DacRedundancy ≠ ∅) then
   conformity = false;
end if;
if (conformity = true) then
   return (conformity);
else
   save_&_return_report ();
   return (non-conformity);
end if;
end.
```

Algorithm 4: Validation of the conformity of concrete access control policies.

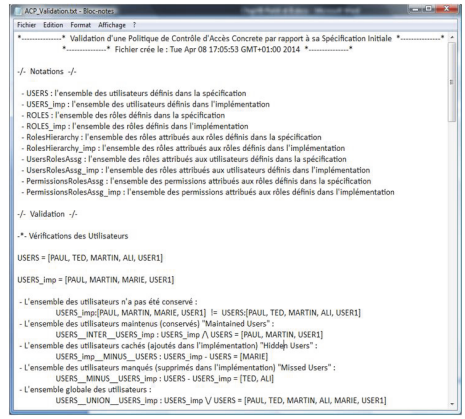


Figure 6: Example of generated report by the validation process.

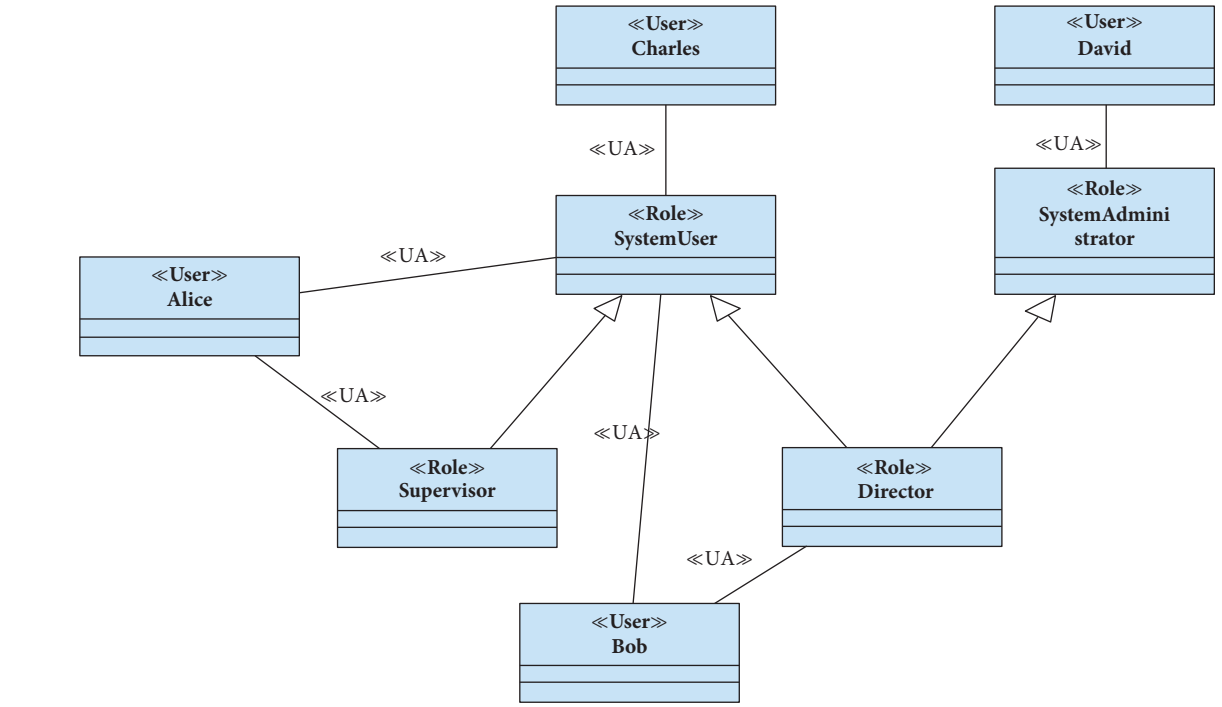
(01) . . .
(02) SETS
(03) USERS = {Bob, David, Alice, Charles};
(04) ROLES = {Director, Supervisor, SystemAdministrator, SystemUser};
(05) OBJECTS = {Meeting, Person, MeetingNotify, MeetingCancel, MeetingModifyStart, MeetingModifyDuration, PersonModifyName};
(06) ACTIONS = {read, create, modify, delete, fullAccess, execute}; . . .
(07) VARIABLES
(08) UsersRolesAssig, RolesHierarchy, PermissionsRolesAssig, . . .
(09) INVARIANT
(10) UsersRolesAssg: USERS − − > POW(ROLES) &
(11) RolesHierarchy: ROLES < − > ROLES &
(12) PermissionsRolesAssig: ROLES − − > (OBJECTS ∗ POW(ACTIONS))& . . .
(13) INITIALISATION
(14) UsersRolesAssg ≔ {(Bob|− > {Director, SystemUser}), (Charles|− > {SystemUser}), (David|− > {SystemAdministrator}), (Alice|− > {Supervisor, SystmUser})} ‖
(15) RolesHierarchy := {(Director|− > SystemAdministrator), (Director|− > SystemUser), (Supervisor|− > SystemUser)} ‖
(16) PermissionsRolesAssig ≔ {(SystemUser|− > (Meeting|− > {create, read})), (SystemUser|− > (Meeting|− > {delete, modify})), (SystemAdministrator|− > (Meeting|− > {read})), (SystemAdministrator|− > (Person|− > {fullAccess})), (Supervisor |− > (Meeting|− > {create, read})), (Supervisor|− > (Meeting|− > {delete, modify})), (Supervisor|− > (MeetingCancel|− > {execute})), (Supervisor|− > (MeetingNotify|− > {execute})), (Director|− > (Meeting |− > {create, read})), (Director|− > (Meeting|− > {delete, modify})), (Director|− > (Meeting |− > {read})), (Director|− > (Person|− > {fullAccess}))} ‖
(17) . . .

Box 14

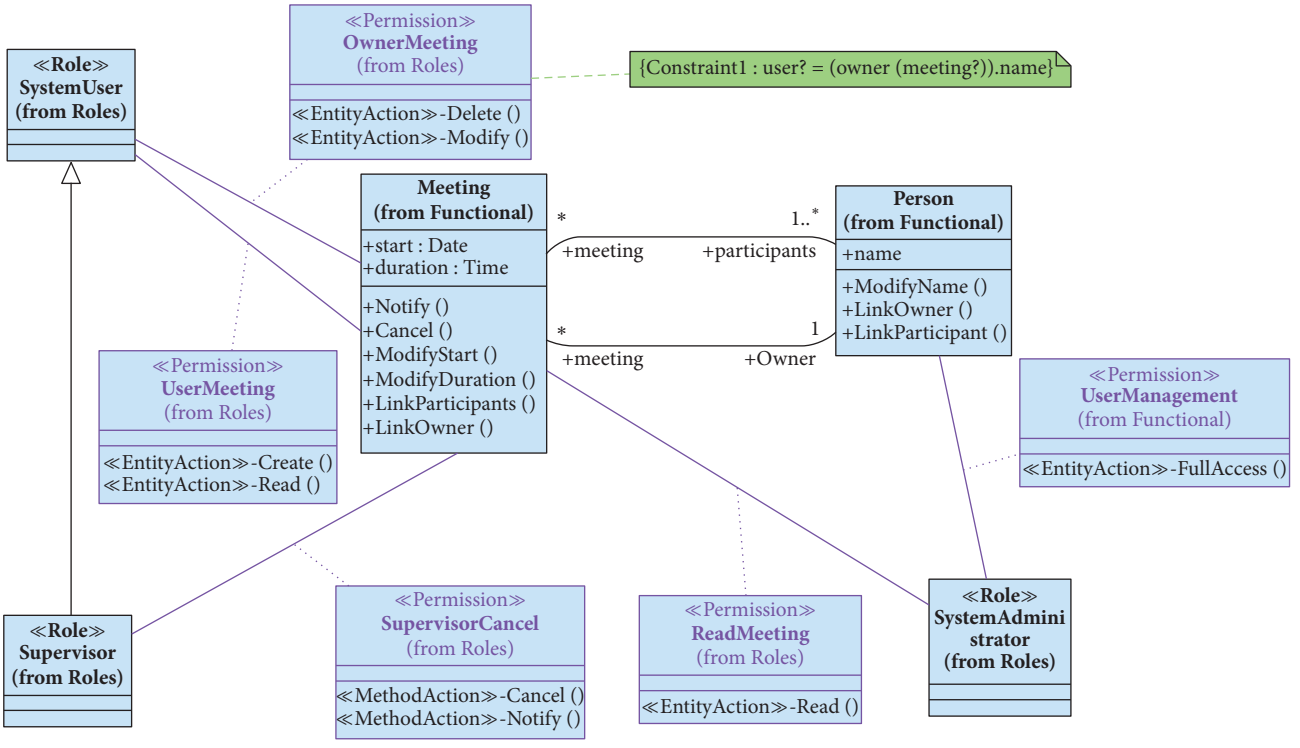
example. This system defines four principal actors. A *system user* is able to create/modify/cancel meetings, add participants to a meeting, and notify the participants about the meeting. The *system administrator* is responsible for managing (creating, modifying information related to, and deleting) persons. The *supervisor* is a special system user, who has the privilege to modify or cancel meetings he does not own. The *director* is both a user and an administrator. A predefined security property requires that a meeting may only be modified/canceled by its owner and supervisors can notify or cancel meetings they do not own.

4.1. Going through Phase 1. The specification of this system defined in Figure 7 describes the assignments of four users (Alice, Bob, Charles, and David) to the corresponding roles (system user, system administrator, supervisor, and director), the hierarchy between roles, and the permissions-roles assignments.

4.2. Going through Phases 2 and 3. This step concerns the formalization of the specified policy. The basic components of the obtained encoding *(instance 1)* of the meeting scheduler access control policy in B notation are as defined in Box 14.



(a) Users-roles and roles-roles assignments



(b) Permissions-roles assignments

Figure 7: Specification of the meeting scheduler with SecureUML.

Now, for the next, and after implementing the valid specification, after a period of time, the access control policy has evolved to a new state where significant changes are introduced.

4.3. Going through Phase 4. This step concerns the extraction of the implemented policy using reverse-engineering techniques. As a result, DDL statements describing the concrete policy are generated and structured as shown in Box 15.

```
(01) CREATE USER ALICE IDENTIFIED BY ALICE DEFAULT TABLESPACE SYSTEM
     TEMPORARY TABLESPACE TEMP;
(02) CREATE USER BOB IDENTIFIED BY BOB;
(03) CREATE USER CHARLES IDENTIFIED BY CHARLIE;
(04) CREATE USER MARIE IDENTIFIED BY MARIE;
(05) CREATE USER PAUL IDENTIFIED BY PAUL;
(06)
(07) CREATE ROLE SYSTEMUSER;
(08) CREATE ROLE SUPERVISOR;
(09) CREATE ROLE SYSTEMADMINISTRATOR;
(10) CREATE ROLE DIRECTOR;
(11) CREATE ROLE COSUPERVISOR;
(12)
(13) GRANT SYSTEMADMINISTRATOR TO DIRECTOR;
(14) GRANT SYSTEMUSER TO DIRECTOR;
(15) GRANT SYSTEMUSER TO SUPERVISOR;
(16) GRANT SUPERVISOR TO COSUPERVISOR;
(17)
(18) GRANT SYSTEMUSER TO ALICE;
(19) GRANT SUPERVISOR TO ALICE;
(20) GRANT SYSTEMUSER TO BOB;
(21) GRANT DIRECTOR TO BOB;
(22) GRANT SYSTEMUSER TO CHARLES;
(23) GRANT SYSTEMADMINISTRATOR TO MARIE;
(24) GRANT COSUPERVISOR TO PAUL;
(25)
(26) GRANT INSERT ON MEETING TO SYSTEMUSER;
(27) GRANT SELECT ON MEETING TO SYSTEMUSER;
(28) GRANT UPDATE ON MEETING TO SYSTEMUSER;
(29) GRANT DELETE ON MEETING TO SYSTEMUSER;
(30) GRANT EXECUTE ON MEETINGCANCEL TO SUPERVISOR;
(31) GRANT EXECUTE ON MEETINGNOTIFY TO SUPERVISOR;
(32) GRANT SELECT ON MEETING TO SYSTEMADMINISTRATOR;
(33) GRANT FULLACCESS ON PERSON TO SYSTEMADMINISTRATOR;
(34) GRANT EXECUTE ON MEETINGCANCEL TO SYSTEMADMINISTRATOR;
(35) GRANT SELECT ON PERSON TO BOB;
```

Box 15

4.4. Going through Phases 5 and 6. The encoding of the extracted policy *(instance 2)* to the target B notation is defined based on an appropriate SQL-B mapping as shown in Box 16.

4.5. Going through Phase 7. The validation process identified the following noncompliance anomalies:

(i) Hidden Users = {Marie, Paul}.

(ii) Missed Users = {David}.

(iii) Renamed Users = ∅.

(iv) Hidden Roles = {Cosupervisor}.

(v) Missed Roles = ∅.

(vi) Renamed Roles = ∅.

(vii) HiddenACFlow = {(Marie|− > {SystemAdministrator}), (Paul|− > {Cosupervisor}), (Cosupervisor|− > Supervisor), (Cosupervisor|− > (Meeting|− > {create, read, delete, modify})), (Cosupervisor|− > (MeetingCancel|− > {execute})), (Cosupervisor|− > (MeetingNotify|− > {execute}))}:

  (a) Hidden ARR = {(Cosupervisor|− > Supervisor)}.

  (b) Hidden AUR = {(Marie|− > {SystemAdministrator}), (Paul|− > {Cosupervisor})}.

  (c) Hidden APR = {(Cosupervisor|− > (Meeting|− > {create, read, delete, modify})), (Cosupervisor|− > (MeetingCancel|− > {execute})), (Cosupervisor|− > (MeetingNotify|− > {execute}))}.

(viii) MissedACFlow = {(David|− > {SystemAdministrator})}:

  (a) Missed ARR = ∅.

  (b) Missed AUR = {(David|− > {SystemAdministrator})}.

  (c) Missed APR = ∅.

(ix) DacRedundancy = {(Bob|− > Director) ∗ ((Person|− > {fullaccess})|− > Director) ∗ (Bob|− > (Person|− > {read}))}.

(x) Redundancy = {(Bob|− > Director) ∗ (Bob|− > SystemUser), (Alice|− > Supervisor) ∗ (Alice|− > SystemUser)}.

```
(01) ...
(02) SETS
(03) USERS_IMP = {Alice, Bob, Charles, Marie, Paul};
(04) ROLES_IMP = {Director, Supervisor, SystemAdministrator, SystemUser, Cosupervisor};
(05) OBJECTS_IMP = {Meeting, Person, MeetingNotify, MeetingCancel,
       MeetingModifyStart, MeetingModifyDuration, PersonModifyName};
(06) ACTIONS_IMP = {read, create, modify, delete, fullAccess, execute}; ...
(07) VARIABLES
(08) UsersRolesAssig_IMP, RolesHierarchy_IMP, PermissionsRolesAssig_IMP, PermissionsUsersAssig_IMP,
(09) INVARIANT
(10) UsersRolesAssg_IMP: USERS_IMP − − > POW(ROLES_IMP) &
(11) RolesHierarchy_IMP: ROLES_IMP < − > ROLES_IMP &
(12) PermissionsRolesAssig_IMP: ROLES_IMP − − > (OBJECTS_IMP ∗ POW(ACTIONS_IMP)) & ...
(13) PermissionsUsersAssig_IMP: USERS_IMP − − > (OBJECTS_IMP ∗ POW(ACTIONS_IMP)) & ...
(14) INITIALISATION
(15) UsersRolesAssg_IMP ≔ {(Bob|− > {Director, SystemUser}), Marie|− >
       {SystemAdministrator}), (Alice|− > {Supervisor, SystemUser}), (Charles|− > {SystemUser}),
       (Paul|− > {Cosupervisor})} ‖
(16) RolesHierarchy_IMP ≔ {Director|− > SystemAdministrator), (Director|− > SystemUser),
       (Supervisor|− >SystemUser), (Cosupervisor|− >  Supervisor)} ‖
(17) PermissionsRolesAssig_IMP ≔ {(SystemUser|− >(Meeting|− > {create, read, delete,
       Modify})), (SystemAdministrator|− >(Meeting|− > {read})), (SystemAdministrator|− >
       (Person|− > {fullAccess})), (Supervisor|− > (Meeting|− > {create, read, delete, modify})),
       (Supervisor|− > (MeetingCancel|− > {execute}), (Supervisor|− > (MeetingNotify|− >
       {execute}), (Cosupervisor|− >(Meeting|− > {create, read, delete, modify})), (cosupervisor|− >
       (MeetingCancel|− > {execute}), (cosupervisor|− > (MeetingNotify|− >
       {execute})),(Director|− > (Meeting|− > {create, read, delete, modify})), (Director|− > (Meeting|− >
       {read})), (Director|− > (Person|− > {fullAccess}))} ‖
(18) PermissionsUsersAssig_IMP ≔ {(Bob|− > (Person|− > {read}))}
(19) ...
```

Box 16

Based on possible interpretations that emphasize for legal changes, the architect updates the specification and/or the implementation to reach the equivalence.

4.6. Going through Phase 8. To simplify the assessment of the risk associated with the detected anomalies, we adopt this hypothesis: the values of the risk associated with the permissions *execute the operation MeetingCancel* and *execute the operation MeetingNotify* are, respectively, evaluated to be 5 and 4 in a scale that varies from 0 to 5. They are considered as the most risky permissions in the system while the rest of the permissions are evaluated to be 1 in the same scale.

Hence, the risk values associated with the detected anomalies are computed and classified according to our defined approach as follows:

(i) R(Hidden Users) = 54.54% *(medium risk)*.

(ii) R(Missed Users) = 15.15% *(minor risk)*.

(iii) R(Renamed Users) = 0% *(minor risk)*.

(iv) R(Hidden Roles) = 43.33% *(medium risk)*.

(v) R(Missed Roles) = 0% *(minor risk)*.

(vi) R(Renamed Roles) = 0% *(minor risk)*.

(vii) R(Hidden ARR) = 71.42% *(high risk)*.

(viii) R(Missed ARR) = 0% *(minor risk)*.

(ix) R(Hidden AUR) = 66.66% *(high risk)*.

(x) R(Missed AUR) = 33.33% *(low risk)*.

(xi) R(Hidden APR) = 25% *(low risk)*.

(xii) R(Missed APR) = 0% *(minor risk)*.

The response monitor detects high risk values associated with hidden assignments of roles to roles and roles to users. It connects to the database and revokes immediately those assignments and sends a report to the *security architect*.

5. Discussion and Conclusion

The emergence of new generation of ICT environments applied to critical domains (such as transportation, communication, finance, and healthcare) generated potential security, reliability, and privacy concerns in the deployment of such critical cyberphysical infrastructures. We explore in this paper the thematic of enhancing security and privacy in critical systems via enhancing the integrity of security policies. We propose a methodology for deploying and monitoring the compliance and managing the life cycle of trusted access control policies. Our methodology extends the traditional life cycle of access control policies with pertinent and necessary activities to manage the compliance of security policies. Moreover, we define a system, *SVIRVRO*, for the practical

application of the defined methodology. The strength of the defined methodology is its generic aspects that make it convenient to the deployment of trusted security policies for critical cyberphysical infrastructures. The application of the defined methodology in the context of small and average system has shown satisfactory and motivating results. We intend to extend the scope of application of this system as discussed in [71].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

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

[2] D. E. Bell and L. J. La Padula, "Secure computer system: Unified exposition and multics interpretation," Defense Technical Information Center, 1975.

[3] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *The Computer Journal*, vol. 29, no. 2, pp. 38–47, 1996.

[4] R. Sandhu, D. Ferraiolo, and R. Kuhn, "NIST model for role-based access control: Towards a unified standard," in *Proceedings of the 5th ACM Workshop on Role-Based Access Control (RBAC)*, pp. 47–63, July 2000.

[5] E. Bertino, G. Ghinita, and A. Kamra, "Access control for databases: Concepts and systems," *Foundations and Trends in Databases*, vol. 3, no. 1-2, pp. 1–148, 2010.

[6] F. Jaidi and F. L. Ayachi, "The problem of integrity in rbac-based policies within relational databases: Synthesis and problem study," in *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication, (ACM IMCOM '15)*, p. 21, Indonesia, January 2015.

[7] F. Jaïdi, F. Labbene-Ayachi, and A. Bouhoula, "Advanced techniques for deploying reliable and efficient access control: Application to E-healthcare," *Journal of Medical Systems*, vol. 40, no. 12, article no. 262, 2016.

[8] F. Jaidi and F. Labbene Ayachi, "A formal system for detecting anomalies of non-conformity in concrete rbac-based policies," in *Proceedings of the International Conference on Computer Information Systems WCCAIS-2015- ICCIS*, p. 2, 2015.

[9] A. Cenys, A. Normantas, and L. Radvilavicius, "Designing role-based access control policies with UML," *Journal of Engineering Science and Technology Review*, vol. 2, no. 1, pp. 48–50, 2009.

[10] T. Lodderstedt, D. Basin, and J. Doser, "SecureUML: A UML-based modeling language for model-driven security," in *Proceedings of the 5th International Conference on The Unified Modeling Language*, pp. 426–441, 2002.

[11] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security: From UML models to access control infrastructures," *ACM Transactions on Software Engineering and Methodology*, vol. 15, no. 1, pp. 39–91, 2006.

[12] J. Jurjens, *Secure Systems Development with UML*, Springer Science & Business Media, 2005.

[13] J. Jürjens, "UMLsec: Exending UML for secure systems development," in *Proceedings of the 5th International Conference on the Unified Modeling Language*, vol. 2460 of *Lecture Notes in Computer Science*, pp. 412–425, Springer, Berlin, Germany.

[14] J. McDermott and C. Fox, "Using abuse case models for security requirements analysis," in *Proceedings of the 15th Annual Computer Security Applications Conference*, pp. 55–64, Phoenix, AZ, USA, 1999.

[15] G. Sindre and A. L. Opdahl, "Capturing security requirements by misuse cases," in *Proceedings of the 14th Norwegian informatics conference*, 2001.

[16] G. Sindre, "Mal-activity diagrams for capturing attacks on business processes," in *Working Conference on Requirements Engineering: Foundation for Software Quality*, pp. 355–366, Springer, Berlin, Germany, 2007.

[17] F. Jaidi and F. Labbene Ayachi, "An approach to formally validate and verify the compliance of low level access control policies," in *Proceedings of the 17th IEEE International Conference on Computational Science and Engineering, CSE 2014 - Jointly with 13th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2014, 13th International Symposium on Pervasive Systems, Algorithms, and Networks, I-SPAN 2014 and 8th International Conference on Frontier of Computer Science and Technology, (FCST '14)*, pp. 1550–1557, China, December 2014.

[18] R. Matulevičius, H. Lakk, and M. Lepmets, "An approach to assess and compare quality of security models," *Computer Science and Information Systems*, vol. 8, no. 2, pp. 447–476, 2011.

[19] R. Matulevicius and M. Dumas, "Towards Model Transformation between SecureUML and UMLsec for Role-based Access Control," *Databases and Information Systems*, no. 3, pp. 339–352, 2011.

[20] E. Karafili, S. Pipes, and E. C. Lupu, "Verification techniques for policy based Systems. IEEE Smart World Congress 2017," in *Workshop on Distributed Analytics InfraStructure and Algorithms for Multi-Organization*.

[21] D. Basin, M. Clavel, J. Doser, and M. Egea, "Automated analysis of security-design models," *Inf. Softw. Technology*, vol. 51, no. 5, pp. 815–831, 2009.

[22] A. Idani, Y. Ledru, J. Richier et al., "Principles of the coupling between UML and formal notations," ANR-08-SEGI-018, 2011.

[23] Y. Ledru, A. Idani, J. Milhau et al., "Taking into account functional models in the validation of IS security policies," *Lecture Notes in Business Information Processing*, vol. 83, pp. 592–606, 2011.

[24] M. Koch, L. V. Mancini, and F. Parisi-Presicce, "A Graph-Based Formalism for RBAC," *ACM Transactions on Information and System Security*, vol. 5, no. 3, pp. 332–365, 2002.

[25] G. Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific Publishing, 1997.

[26] M. Nyanchama and S. Osborn, "The role graph model and conflict of interest," *ACM Transactions on Information and System Security*, vol. 1, no. 2, pp. 3–33, 1999.

[27] R. W. Baldwin, "Naming & grouping privileges to simplify security management in large databases," in *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pp. 116–132, Los Alamitos, Calif, USA, 1990.

[28] N. Zhang, M. Ryan, and D. P. Guelev, "Evaluating access control policies through model checking," in *Information Security Journal*, vol. 3650 of *Lecture Notes in Computer Science*, pp. 446–460, Springer, Berlin, Germnay, 2005.

[29] S. Kikuchi, S. Tsuchiya, M. Adachi, and T. Katsuyama, "Policy verification and validation framework based on model checking approach," in *Proceedings of the 4th International Conference on Autonomic Computing, (ICAC'07)*, pp. 1–9, USA, June 2007.

[30] M. Bartoletti, P. Degano, G. . Ferrari, and R. Zunino, "Model checking usage policies," *Mathematical Structures in Computer Science*, vol. 25, no. 3, pp. 710–763, 2015.

[31] S. Ranise, A. Truong, and R. Traverso, "Parameterized model checking for security policy analysis," *International Journal on Software Tools for Technology Transfer*, vol. 18, no. 5, pp. 559–573, 2016.

[32] A. Armando and S. Ranise, "Scalable automated symbolic analysis of administrative role-based access control policies by SMT solving," *Journal of Computer Security*, vol. 20, no. 4, pp. 309–352, 2012.

[33] S. Ranise, A. Truong, and A. Armando, "Boosting model checking to analyse large ARBAC policies," *STM 2012*, vol. 7783, pp. 273–288, 2013.

[34] H. Zhao, J. Lobo, and S. M. Bellovin, "An algebra for integration and analysis of Ponder2 policies," in *Proceedings of the 9th IEEE Workshop on Policies for Distributed Systems and Networks, (POLICY '08)*, pp. 74–77, USA, June 2008.

[35] C. Basile, A. Cappadonia, and A. Lioy, "Network-level access control policy analysis and transformation," *IEEE/ACM Transactions on Networking*, vol. 20, no. 4, pp. 985–998, 2012.

[36] F. Turkmen, J. Den Hartog, S. Ranise, and N. Zannone, "Analysis of XACML policies with SMT," *POST, ETAPS*, vol. 9036, pp. 115–134, 2015.

[37] A. Armando and S. Ranise, "Automated and efficient analysis of role-based access control with attributes," *DBSec*, vol. 7371, pp. 25–40, 2012.

[38] P. Gupta, S. D. Stoller, and Z. Xu, "Abductive analysis of administrative policies in rule-based access control," *ICISS*, vol. 7093, pp. 116–130, 2011.

[39] H. Kamoda, M. Yamaoka, S. Matsuda, K. Broda, and M. Sloman, "Policy conflict analysis using free variable tableaux for access control in web services environments," *Policy Man. for the Web*, vol. 2, pp. 5–12, 2005.

[40] F. Hansen and V. Oleshchuk, "Conformance checking of RBAC policy and its implementation," in *Information Security Practice and Experience*, vol. 3439 of *Lecture Notes in Computer Science*, pp. 144–155, Springer, Berlin, Germany, 2005.

[41] C. Huang, J. Sun, X. Wang, and Y. Si, "Security policy management for systems employing role based access control model," *Information Technology Journal*, vol. 8, no. 5, pp. 726–734, 2009.

[42] R. Thion and S. Coulondre, "A relational database integrity framework for access control policies," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 131–159, 2012.

[43] A. Ghadi, *Modèle hiérarchique de contrôle d'accès d'UNIX basé sur un graphe de rôles*, 2010.

[44] R. Laleau and A. Mammar, "An overview of a method and its support tool for generating B specifications from UML notations," in *Proceedings of the 15th IEEE International Conference on Automated Software Engineering,)ASE '00)*, pp. 269–272, France, September 2000.

[45] K. Lano, D. Clark, and K. Androutsopoulos, "UML to B: Formal verification of object-oriented models," *Integrated Formal Methods, Lecture Notes in Computer Science*, vol. 2999, pp. 187–206, 2004.

[46] J. Milhau, A. Idani, R. Laleau, M. A. Labiadh, Y. Ledru, and M. Frappier, "Combining UML, ASTD and B for the formal specification of an access control filter," *Innovations in Systems and Software Engineering*, vol. 7, no. 4, pp. 303–313, 2011.

[47] C. Snook and M. Butler, "U2B - A tool for translating UML-B models into B," in *UML-B Specification for Proven Embedded Systems Design*, Springer, Berlin, Germany, 2004.

[48] "B4Msecure tool," http://b4msecure.forge.imag.fr/.

[49] C. Soutou, "Outils d'aide à la conception de bases de données, une synthèse" (French).

[50] S. Martínez, V. Cosentino, J. Cabot, and F. Cuppens, "Reverse engineering of database security policies," in *Proceedings of the 24th International Conference on Database and Expert Systems Applications*, vol. 8056, pp. 442–449.

[51] A. Mammar, *Un environnement formel pour le développement dapplications base de données*, 2002.

[52] H. P. Nguyen, *Dérivation de spécifications formelles B à partir de spécifications semiformelles*, CNAM, 1998 (French).

[53] S. Chakraborty and I. Ray, "Integrating trust relationships into the rbac model for access control in open systems," in *Proceedings of the 11th ACM symposium on Access control models and technologies*, pp. 49–58, June 2006.

[54] F. Fujun, L. Chuang, P. Dongsheng, and L. Junshan, "A Trust and Context based access control model for distributed systems," in *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications, (HPCC '08)*, pp. 629–634, China, September 2008.

[55] B. Bhargava and L. Lilien, "Vulnerabilities and threats in distributed systems," in *Distributed Computing and Internet Technology*, vol. 3347 of *Lecture Notes in Computer Science*, pp. 146–157, Springer, Berlin, Germany, 2005.

[56] D. Ferraiolo, J. Cugini, and D. R. Kuhn, "Role-based access control (RBAC): Features and motivations," in *Proceedings of the 11th IEEE Annual Computer Security Application Conference*, pp. 241–248, 1995.

[57] R. T. Simon and M. E. Zurko, "Separation of duty in role-based environments," in *Computer Security Foundations Workshop*, pp. 183–194, 1997.

[58] V. D. Gligor, S. I. Gavrila, and D. Ferraiolo, "On the formal definition of separation-of-duty policies and their composition," *Security & Privacy*, 1998.

[59] T. Jaeger, "On the increasing importance of constraints," in *Proceedings of the the fourth ACM workshop on Role-based access control*, pp. 33–42, Fairfax, Virginia, USA, October 1999.

[60] L. Chen and J. Crampton, "Risk-Aware Role-Based Access Control," in *Proceedings of the 7th International Workshop on Security and Trust Management*, vol. 7170, 2011.

[61] P.-C. Cheng, P. Rohatgi, C. Keser, P. A. Karger, G. M. Wagner, and A. S. Reninger, "Fuzzy multi-level security: an experiment on quantified risk-adaptive access control," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '07)*, pp. 222–230, IEEE, Berkeley, Calif, USA, May 2007.

[62] Q. Ni, E. Bertino, and J. Lobo, "Risk-based access control systems built on fuzzy inferences," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communication Security (ASIACCS '10)*, pp. 250–260, New York, NY, USA, April 2010.

[63] I. Molloy, L. Dickens, C. Morisset, P. Cheng, J. Lobo, and A. Russo, "Risk-based security decisions under uncertainty," in *Proceedings of the the second ACM conference*, pp. 157–168, San Antonio, Texas, USA, Feburary 2012.

[64] J. Ma, K. Adi, M. Mejri, and L. Logrippo, "Risk analysis in access control systems," in *Proceedings of the 8th International Conference on Privacy, Security and Trust, (PST '10)*, pp. 160–166, Canada, August 2010.

[65] N. Nissanke and E. J. Khayat, "Risk based security analysis of permissions in RBAC," in *Proceeding of the 2nd International Workshop on Security In Information Systems, INSTICC Press*, pp. 332–341, 2004.

[66] J. R. Abrial, *The B-Book: Assigning Programs to Meanings*, Press Syndicate of the University of Cambridge, Cambridge, UK, 2005.

[67] "ClearSy: B Language Reference Manual, v.1.8.6," 2007.

[68] F. Jaidi and F. Labbene Ayachi, "A reverse engineering and model transformation approach for RBAC-administered databases," in *Proceedings of the 13th International Conference on High Performance Computing & Simulation*, 2015.

[69] F. Jaidi and F. L. Ayachi, "A formal approach based on verification and validation techniques for enhancing the integrity of concrete role based access control policies," *Advances in Intelligent Systems and Computing*, vol. 369, pp. 53–64, 2015.

[70] F. Jaidi and F. L. Ayachi, "A risk awareness approach for monitoring the compliance of RBAC-based policies," in *Proceedings of the 12th International Conference on Security and Cryptography, (SECRYPT '15)*, pp. 454–459, July 2015.

[71] F. Jaidi and F. Labbene Ayachi, "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.