Declassification Policy Management in Dynamic Information Systems

Julien A. Thomas, Nora Cuppens-Boulahia, Frédéric Cuppens
Télécom Bretagne ; LUSSI Department
Université Européenne de Bretagne
Cesson Sévigné, France

Abstract—Standard multilevel security (MLS) policies lack flexibility as data classification is considered static. Previous works have addressed this issue and defined declassification requirements, especially in programming languages using a language-based security approach. In this paper, we suggest a different approach. We show how to define and enforce declassification policies in databases, seen as sets of logical facts. We first define an information flow control model where data classification may dynamically change. This model combines both confidentiality and integrity requirements to enforce security. We then specify how to enforce declassification policies. Our approach relies on Event-Condition-Action (ECA) rules and provides means to manage the four basic dimensions of declassification, namely the what?, who?, where? and when? which respectively refer to modeling information to be declassified, entities responsible for declassification, localization of the declassification and contextual conditions that control declassification. We formalize and specify our declassification policies and prove it safe and secure with respect to the information flow control model.

I. INTRODUCTION

Many research works like [23], [6], [7], [9] address the issue of modeling multilevel database security policies. However, most of these works assume that data are assigned static classifications (the so called tranquility principle). This is a restricted hypothesis since data initially inserted at the secret level may become public. Several works like [14] followed by [21], [13], [17], [19], [20], [15] investigated declassification requirements using language-based security. In these works, mechanisms were defined to analyze whether security requirements are correctly implemented. However, managing declassification securely is a complex issue for language-based security since we aim at authorizing information flow that would be normally considered illegal in classical confidentiality settings like non-interference [14].

In this paper, we investigate the declassification issue in the context of databases, seen as a set of facts of a logical theory. This modeling provides a higher view of data than programming languages and we argue that this higher view provides the appropriate level to “semantically” manage declassification. In particular, we show how to specify declassification policies that may depend on human decision and sophisticated contextual conditions. We formally specify how to manage such declassification policies using Event-Condition-Action (ECA) rules [10], which may be viewed as a generalization of the trigger mechanisms implemented in many information systems. Since declassification operations raise specific security problems due to new information flows, it is necessary to extend traditional MLS models, like non-interference, to handle them. One difficulty is that we need to guarantee that the declassified data is actually compliant with the declassification policy. More precisely, a malicious user may actually attempt to insert or modify some data in order to take advantage of the declassification process to illegally declassify data that should remain secret. For example, let us consider the following downgrading rule: By the end of the war, invasion objectives must be declassified. A malicious user may attempt to falsely insert that the war is over to illegally trigger the declassification of the secret invasion objectives. Thus, declassification not only raises confidentiality but also integrity issues and we need to define a robust information flow control model. This model differs from previous studies and combines an integrity policy with a safe and secure declassification policy. We then show and prove how this model can be enforced in databases using the ECA rule paradigm.

Our paper is organized as follows. In Section II we present the concept of active information systems and declassifications and introduce relevant security issues. In Section III we define a security model which considers both integrity and confidentiality requirements. This model formalizes the associated security model to control both information flows created by declassification and by active rule execution. In Section IV we show how to enforce this security model in information systems. We also specify declassification policies that may depend on contextual conditions, based on ECA rules. In Section V, we present related works. Finally, Section VI concludes and presents future work.

II. ACTIVE INFORMATION SYSTEMS AND DECLASSIFICATION ISSUES

In this section, we first present the concept of declassification. We then introduce our model for active
information system and declassification policy. In order to pinpoint several issues associated with declassification policy in active information systems, we finally present several examples of insecure declassifications.

A. Declassification Policy

When considering multilevel systems, a clearance level is assigned to subjects (the active entities that includes users) and a classification level is assigned to objects (the passive entities representing information to be protected). Several studies consider that this classification level assigned to objects is static and may not be modified by user queries as this is part of the administration policy. However, such an assumption, called the tranquility principle, reduces the expressiveness of the security policy. More recently, the modification of object classifications by user queries as this is part of the administration policy is defined, we then need to specify an information flow model which controls that declassification operations do not create illegal information flow. This model is defined in Section III. In Section IV we finally specify practical requirements to enforce the information flow control model into an information system and we prove that these requirements are sufficient conditions.

B. Information System Model with Dynamic Behavior

We adopt a logical representation of the information system. It consists of active entities (USERS), operations (OPS) and system states (STATES). The operations are the select, delete and insert operations. The update operation is modeled as a sequence of a delete operation followed by an insert operation. The system states are modeled through a set predicates (PREDICATES). The system state then corresponds to a set of fully instantiated predicates pi called facts. We consider a valuation function II: (PREDICATES \times STATES) → BOOL. We say that a fact pi holds in state s if II(p, s) = True.

To model information systems with dynamic behavior, we use the ECA-rule paradigm defined in [10]. This paradigm was extended by other works [11], [2]. In [2], Baral, Lobo and Trajcevski propose the Lactive language. An application of this language is the modeling of active databases [12].

1) Actions: The first dimension is the action definition. In Lactive, we thus consider a set ACTIONS. The definition of actions rules do(\alpha) is modeled as do(\alpha) causes op1, ..., opk if q1(X1), ..., qn(Xn); where op1, ..., opk is a sequential execution of select, delete or insert operations and q1(X1), ..., qn(Xn) is the conjunctive pre-condition to be satisfied to execute the action.

2) Events: As we consider active databases, actions may trigger events belonging to a set of EVENTS. Events are defined as follows: event(X) after do(\alpha) if t1(X1), ..., tm(Xm);

3) Active Rules: Finally, the occurrence of events may in turn initiate the execution of a sequence of new actions though an active rule (belonging to a set of RULES). An active rule is defined as: rule : event(X) initiates do(\alpha1)...do(\alphak) if t1(X1), ..., tp(Xp);

In Section II-D we explain how to use the Action, Event and Active rule paradigms to specify application dependent security policies. Before, we present the concepts we need to model a multilevel security policy.

C. MultiLevel Security

To model a Multilevel Security Policy, we first add a set LEVELS of security levels and the inf_level, equal_level and inf_equal_level relations. These relations are defined on (LEVELS \times LEVELS) → BOOL. Regarding the last relation, inf_equal_level(lvl1, lvl2) is a transitive and reflexive relation, abbreviated as
level1 ⊆ level2 in the following. For example, we can consider the security levels TS (Top Secret), S (Secret) and P (Public or ⊥), with P ⊆ S ⊆ TS. We also define functions to assign security levels to subjects and objects: clearance_level ∈ USERS → LEVELS, and classification_level ∈ PREDICATES → LEVELS.

Finally, we require the security policy to satisfy the known policy constraint, which means that the multilevel security policy definition is itself unclassified. In order to meet this requirement, the values of the security functions must be classified at the lowest level denoted ⊥. This prevents the classification_level information to flow while consulting the policy, as it is classified at the lowest level ⊥ and already known by all users. We also assume that clearance_level may not be modified by user queries. This assumption is done as we consider such queries to be part of the policy administration.

D. Declassification policy specification

In our work, we use the \( L_{\text{active}} \) language to specify application dependent declassification policies. A declassification of a piece of information corresponding to a fact \( p \) is modeled as an action that updates the classification level of this fact at a lower level. This is modeled by the action \( \text{declassify}(P, L) \) where \( P \) is a fact to be declassified and \( L \) is the new classified level of this fact. Using \( L_{\text{active}}, \) \( \text{declassify}(P, L) \) is functionally specified as the following action:

\[
\text{do}(\text{declassify}(P, L)) \quad \text{causes} \quad \text{delete}(\text{classification}_\text{level}(P, L')) , \quad \text{insert}(\text{classification}_\text{level}(P, L)) \quad \text{if} \quad \text{classification}_\text{level}(P, L') \wedge \text{inf}_\text{level}(L, L')
\]

Of course, this corresponds to functional specification of declassification and other requirements, especially information flow controls, are necessary to enforce secure declassification as specified in Section II below.

1) Trust based downgrading specification: As suggested in Section II-A, trust based downgrading is modeled as an access control requirement. For this purpose, we consider the predicate is_permitted defined on USERS \( \times \) ACTIONS. If \( u \) is a user and \( \alpha \) is an action, then is_permitted(\( u, \alpha \)) means that \( u \) is permitted to execute action \( \alpha \). Thus to specify trust based downgrading requirements, we have to specify which declassification actions each user is permitted to execute. For example, if warCost(C) is a predicate which specifies the total cost of a war, then is_permitted(john, declassify(warCost(C), S)) means that user John is permitted to declassify predicate warLocation(C) at level S.

2) Automatic downgrading specification: Automatic downgrading is modeled through an active rule which automatically triggers a downgrading operation when some events occur. For this purpose, we first define the event \( \text{declassify}_\text{event}(P, L) \). When this event occurs, the specified automatic declassification occurs. This is modeled by the following active rule:

\[
\text{declassify_rule:} \quad \text{declassify}_\text{event}(P, L) \quad \text{initiates do}(\text{declassify}(P, L)) ;
\]

Specifying automatic downgrading then consists in specifying when \( \text{declassify}_\text{event}(P, L) \) occurs. For example, let us consider the automatic downgrading requirement: the total war cost is automatically downgraded at the public level when victory occurs. This requirement is specified by the \( \text{declassify}_\text{event} \) event:

\[
\text{declassify}_\text{event}(\text{warCost}(C), P) \quad \text{after do}(\text{insert}(\text{victory})) \quad \text{if} \quad \text{warCost}(C) ;
\]

This corresponds to an application dependent downgrading requirement.

3) Declassification Policy Consistency: In the remainder of this paper, we assume that the declassification policy is correctly specified, in particular it is consistent. To enforce the declassification policy consistency, one may observe that declassifying some sensitive information may entail the declassification of other sensitive information that may be derived from the declassified information. This problem is closely related to the so-called inference problem in multilevel information systems. However, this issue is out of the scope of this paper and is left for future work. So, in the following, we focus on controlling information flow created by declassification and we consider that possible dependencies between information is correctly handled within the declassification policy.

E. Needs of integrity controls for secure declassification

At first glance, controlling declassification looks like a simple problem. It seems sufficient to consider that information flow created by declassification of some information at a given level \( L \) becomes authorized information flow after the declassification occurs. However and as mentioned before [17], [24], it is important to consider the integrity level of the dependencies, in order to define secure declassification operations. In this section, we illustrate why this concept of integrity is important.

Consider the declassification policy defined in Section II-D. According to this declassification policy, a user with a clearance level \( S \) may attempt to update the warLocation(C) predicate classified at TS with other sensitive information classified at TS, for example information number_of_death(Nb) representing the number of deaths during the war. This would correspond to blind write up that would not create illegal information flow. However, when the predicate warCost(C) is declassified, the cost value would actually correspond to the number of deaths. Thus declassifying warCost(C)
is responsible for the indirect insecure declassification of number_of_death(Nb). This first example illustrates the illegal declassification of sensitive data using blind write up. In order to avoid this type of issues, the integrity of the information has to be considered when managing declassification operations.

Second, automatic downgrading depends on contextual events whose occurrence triggers declassification. Consider the example of automatic downgrading presented in Section [H-I-D]. In this example, declassification of warCost(C) is automatically triggered when the predicate victory is inserted in the database. Thus, a user with a clearance lower than TS may attempt to illegally insert victory to trigger the declassification of warCost(C). Thus we must also control the integrity of triggering events to prevent this type of insecure flows.

Due to these observations, it is obvious that the integrity of data is important. In the following, we shall consider that information are assigned both confidentiality and integrity levels. It is generally considered that confidentiality and integrity levels take their values in different sets of security levels. However, when controlling declassification, this is not appropriate as we aim at controlling that information classified L have not been illegally modified by actions or events classified lower than L. Thus, we consider the integrity levels take their value in the same set as confidentiality levels.

Notice also that we assume in the following that a user cleared at a given level L is trusted and would not attempt to illegally declassify information classified L. Thus, from the point of view of controlling declassification, every action performed by a user cleared at level L has an integrity level equal to L.

III. Security Models

In Section [H-I-E] we suggest modeling declassification policies using ECA rules. In this section, we define an information flow control model to securely manage such declassification policies. Of course, standard information flow control policies are not appropriate to manage declassification policies. However, ECA rules also create specific information flow so that we first need to adapt existing information flow control models. Thus, we first specify in this section the information flow control models for standard information systems without ECA rules, then for active information systems with ECA rules and finally for active systems with declassification.

A. Information Systems without declassification

Our model relies on the trace based security formalism defined in [5]. We consider a set TRACES of traces defined as follows:

- A trace \( t = \{s_0, \alpha\} \) is composed of the initial state \( s_0 \) and an infinite sequence \( \alpha \) of actions executed in trace \( t \).
- If \( t = \{s_0, \alpha\} \) is a trace, then \( t_n = \{s_0, \alpha_n\} \) is a finite trace composed of the initial state \( s_0 \) and the sequence \( \alpha_n \) of the first actions executed in \( t \).
- If \( t_n = \{s_0, \alpha_n\} \) is a finite trace, then \( s_n \) is the state obtained after executing the sequence of actions \( \alpha_n \) starting from the initial state \( s_0 \).
- We assume that each action in a trace is executed at a given security level. If \( \alpha \) is a sequence of actions and \( L \) is a security level, then \( \alpha[L] \) is the subsequence of actions whose security level is lower than or equal to \( L \).
- Two states \( s \) and \( s' \) are equivalent at level \( L \) (denoted \( s \approx_L s' \)) if (1) the classification of predicates classified lower than or equal to \( L \) is identical in \( s \) and \( s' \) and (2) the truth value of predicates classified lower than or equal to \( L \) is identical in \( s \) and \( s' \).
- Equivalence between traces at same security level \( L \) is defined as follows: \( t \sim_L t' \iff s_0 \approx_L s'_0 \land \alpha[L] = \alpha'[L] \). This definition says that two traces are indistinguishable at \( L \) if the initial states and the subsequence of actions whose security level is lower than or equal to \( L \) are identical in \( t \) and \( t' \).

Now, in order to define our security properties, we consider two traces \( t_n = \{s_0, \alpha_n\} \) and \( t'_n = \{s'_0, \alpha'_n\} \). In standard databases, the confidentiality property at level \( L \) is defined as the equivalence at level \( L \) between the state of the systems when the considered traces are indistinguishable at level \( L \):

\[
1) \forall(t, t'n, n'), (t, t' : TRACES, n, n' : INTEGER) \Rightarrow (t_n \sim_L t'_n \Rightarrow s_n \approx_L s'_n)
\]

In active databases, however, traces also contain the triggered actions. We refer to the subset of queries in \( \alpha \) issued by the users by ext(\( \alpha \)) (user external actions). The database states must only depend on the user queries and the initial database states. For this purpose we introduce the following notation:

- \( \text{ext}(t) \sim_L \text{ext}(t') \iff s_0 \approx_L s'_0 \land \text{ext}(\alpha[L] = \text{ext}(\alpha')[L] \)

Using this notation, for every trace \( t_n \) and \( t'_n \), the confidentiality property is then defined as the equivalence of the traces and the database states when the external actions are equivalent:

\[
1) \text{ext}(t_n) \sim_L \text{ext}(t'_n) \Rightarrow \alpha_n[L] = \alpha'_n[L] \\
2) t_n \sim_L t'_n \Rightarrow s_n \approx_L s'_n.
\]

The second property is identical to the one required for standard databases. The first property requires that the sequence of actions executed at level \( L \) in a given trace should be determined by the sequence of user external actions executed at level \( L \) and information classified at level \( L \) in the initial state. As shown in [5],
actions are modeled as declassify(Predicate, L) in the traces.

- If $\alpha$ is a sequence of actions and $L$ is a security level, then $\alpha[L_D]$ is the subsequence of actions having the form $\text{declassify}(\text{Predicate}, L)$ or actions whose security level is lower than or equal to $L$.
- If $\alpha_k$ is the $k^{th}$ action of a sequence of actions $\alpha$, then the function $\text{indexof}(\alpha_k, \alpha)$ represents the position of $\alpha_k$ in the trace $\alpha$, namely $\text{indexof}(\alpha_k, \alpha) = k$.
- Two traces $t = (s_0, \alpha)$ and $t' = (s_0, \alpha')$ are equivalent from the point of view of information declassified at level $L$ (denoted $t \approx_{L_D} t'$) if:
  1) The $\alpha_D$ subsequence of declassification actions at level $L$ is identical in traces $t$ and $t'$ and
  2) For every declassification action $d = \text{declassify}(P, L)$, if $\text{indexof}(d, \alpha) = i$ and $\text{indexof}(d, \alpha') = i'$ then $\Pi(P, s_i) = \Pi(P, s_{i'})$

The second condition says that the truth value of declassified information is identical in $t$ and $t'$.

- We then modify equivalence relation between traces at some security level $L$ when declassification operations are considered as follows: $t \approx_{L_D} t' \Leftrightarrow s_0 \approx_L s_0' \wedge \alpha[L_D] = \alpha'[L_D] \wedge t \approx_{L_D} t'$.

1) Confidentiality in Information Systems with Declassification: As mentioned in Section I-D, we consider two different types of declassification: Trust based downgrading and automatic downgrading. Trust based downgrading corresponds to external actions whereas automatic downgrading corresponds to event triggered actions. Then we define the confidentiality property with declassification for every trace $t_n$ and $t'_n$, as:

1. $\text{ext}(t_n) \sim_L \text{ext}(t'_n) \Rightarrow \alpha_n[L_D] = \alpha'_n[L_D]$  
2. $t_n \sim_{L_D} t'_n \Rightarrow s_n \approx_{L_D} s'_n$.

The first property specifies that the occurrence of event triggered actions at level $L$ (including automatic declassification) should be determined by authorized knowledge of users at level $L$. Here authorized knowledge includes information declassified by trust based downgrading actions.

The second property specifies that the system state at level $L$ should be determined by authorized knowledge of users at level $L$. However, once the occurrence of automatic declassification at level $L$ is determined by authorized knowledge of users at level $L$, then the effect of such automatic downgrading can be included in the authorized knowledge of users at level $L$, as we assume that the declassification policy is correctly specified.

2) Integrity in Information Systems with Declassification: As explained in Section II-E, we must also consider the integrity of the declassification operations. For this purpose, let us assume that some information classified at level $L$ is declassified at a lower level $L'$. The integrity property must guarantee that this information corresponds to information “really” classified at level $L$, i.e. this information has not been modified by actions classified at a level lower than $L$. For this purpose, we introduce the following definitions:

- If $\alpha$ is a sequence of actions and $L$ is a security level, then $\alpha[L]$ is the subsequence of actions whose security level is $L$. The second property specifies that the system state at level $L$ is identical in traces $t$ and $t'$.
- If $s$ and $s'$ are states then $s \approx_L s'$ iff (1) the classification of predicates whose classification is equal to $L$ is identical in $s$ and $s'$ and (2) the truth value of predicates whose classification is equal to $L$ is identical in $s$ and $s'$.
- We then modify equivalence relation between traces at some security level $L$ when declassification operations are considered as follows: $t \approx_{L_D} t' \Leftrightarrow s_0 \approx_L s_0' \wedge \alpha[L_D] = \alpha'[L_D] \wedge t \approx_{L_D} t'$.

Using these definitions, we define the integrity property for declassified information for every trace $t_n$ and $t'_n$, as:

- $\text{ext}(t_n) \sim_L \text{ext}(t'_n) \Rightarrow t_n \approx_{L_D} t'_n$.

This property says that declassified information coming from classification level $L$ should be strictly determined by information initially classified at level $L$ or by external actions classified at level $L$. Notice that event triggered actions are not included in the condition since they may be activated by events classified at a level lower than $L$. However, this property does not take into account the case where information initially classified at level $TS$ are first declassified at level $S$ and then $P$. For this purpose, we refine the above definition:
• $\text{ext}(t_n) \sim_L \text{ext}(t'_n) \land t_n \not\sim_L t'_n \Rightarrow t_n \approx_{DL} t'_n$.

This property says that declassified information coming from classification level $L$ should be strictly determined by information initially classified at level $L$, by external actions classified at level $L$ or by information declassified at level $L$. Of course, these definitions are interdependent: declassifications from level $S$ to $P$ are secure if declassifications from level $TS$ to $S$ are also secure.

IV. SECURITY PROPERTY ENFORCEMENT IN INFORMATION SYSTEMS WITH DECLASSIFICATION

In Section III we presented security requirements to control information flow in multilevel information systems with declassification functionalities. In this section, we show how to enforce these security requirements.

A. ECA Rules Security Model

As shown in Section III we need to control both confidentiality and integrity of information to securely manage declassification. For this purpose, we define a function to assign integrity levels to facts: $\text{integrity}_\text{level} \in PREDICATES \rightarrow LEVELS$. As for the confidentiality policy, we require the integrity policy to satisfy the known policy constraint.

Multilevel Action: We extend the do operator by adding two parameters which respectively represent the confidentiality and integrity levels of the executed action. Thus, a multilevel action is defined with the do operator on $ACTIONS \times LEVELS \times LEVELS$: $\text{do}(\alpha, Lc, Li) \text{ causes } op_1 \ldots op_k \text{ if } q_1(X_1), \ldots, q_n(X_n)$.

Multilevel Event: The second definition is the event definition which extends the previous definition with security level parameters. A multilevel event is defined as $\text{event}_\text{name}(X, Lc_e, Li_e) \text{ after } \text{do}(\alpha, Lc, Li) \text{ if } r_1(Z_1) \ldots, r_m(Z_m)$.

Multilevel Active Rule: Finally, we revise the active rule definition by adding security levels. A multilevel active rule is modeled by the following template: $\text{rule}_\text{name} : \text{event}_\text{name}(X, Lc_e, Li_e) \text{ initiates } \text{do}(\alpha_1, Lc_1, Li_1) \ldots \text{do}(\alpha_j, Lc_j, Li_j) \text{ if } t_1(Z_1) \ldots, t_p(Z_p)$.

B. Confidentiality Policy

In order to enforce confidentiality, we define the Access Law and Modification Law to respectively control access to and modification of the database. We finally define the User Privilege Law for each user query.

Access Law: If $p$ is a predicate and $l$ a security level, the Access Law is satisfied w.r.t. $(p, l)$ if the classification level of $p$ is lower than or equal to $l$.

Modification Law: If $p$ is a predicate and $l$ a security level, the Modification Law is satisfied w.r.t. $(p, l)$ if $l$ is lower than or equal to the classification of $p$.

User Privilege Law: If $s$ is a user and $l$ is a security level, the User Privilege Law is satisfied w.r.t. $(s, l)$ if $l$ is the clearance level of $s$.

C. Integrity Policy

Our objective is to control declassification by evaluating the integrity of the information. The enforcement conditions assert that, after an interaction, the integrity levels of the targets (triggered actions and events) are lower than the integrity levels of the sources (parent event and action). We then say that the security policy only allows (trusted) actions to modify an object:

Integrity Evaluation Law: The integrity of any action, event, rule or object depends on the integrity of the entities which triggered it so that the integrity is always decreasing throughout the dependency chain.

Integrity Policy: Let us consider a modification (insert, delete, update) request performed with an integrity level $Li$. An object $obj$ may be modified if the integrity $Ls$ of the object satisfies $Ls \subseteq Li$.

Due to the Integrity Evaluation law which enforces that the action integrity level is at most equal to the lower bound of the conditions the actions rely on, the integrity policy enforces the following integrity requirement presented in Section III-D:

$\text{ext}(t_n) \sim_L \text{ext}(t'_n) \land t_n \not\sim_L t'_n \Rightarrow t_n \approx_{DL} t'_n$.

1) Integrity Declassification Policy: As stated in the previous section, we associate the write policy with a strict integrity policy, in order to prevent uncontrolled modification of the integrity of objects. This allows us to prevent untrusted users from blocking trusted declassifications. However, in order to lower the limitations of the strict integrity policy and still allow the decrease of the integrity levels, we also consider an integrity declassification policy which is defined as follows: “a user $U$ is permitted to declassify the integrity level of an object $O$ from $LO$ to $L_d$ if such declassification is explicitly authorized by the security policy”. When considering the $\text{is permitted}$ predicate presented in Section III-D this is specified by $\text{is permitted}(U, \text{declassify}\_\text{integrity}(P, L_o))$.

For our concern, users must not prevent data from being declassified due to untrusted declassifications of integrity levels. Indeed, this would produce issues similar to decreasing the integrity levels [4]. Our declassification policy must thus not rely on the standard low-watermark policy for objects where $\forall(U, P, L)$. $(\text{is permitted}(U, \text{declassify}\_\text{integrity}(P, L)), \text{as any user may decrease the integrity of sensitive data and thus block any authorized declassifications. In order to prevent unauthorized users from blocking declassification actions, we require users authorized to declassify the integrity of an object to also be authorized to declassify the object: $\text{is permitted}(U, \text{declassify}\_\text{integrity}(P, L_o)) \Rightarrow \text{is permitted}(U, \text{declassify}\_\text{integrity}(P, L_d))$. In order to enforce this specification of the integrity declassification policy, we define the following active rule associated to the action $\text{declassify}\_\text{integrity}$.
the predicate \( \text{integrity}_\text{level} \) being classified \( \perp \), such an action must perform insert and delete operations at \( \perp \). Thus, the integrity declassification action must be either ordered by a user or triggered by a public event:

\[
\text{do}(\text{declassify}_\text{integrity}(U,P,L), \perp, L') \text{ causes } \text{delete}(\text{integrity}_\text{level}(P,L'), \text{insert}(\text{integrity}_\text{level}(P,L))
\]

\[
\text{if } \text{integrity}_\text{level}(P,L') \land \inf\text{level}(L',L) \land \text{is}_\text{permitted}(U,\text{declassify}_\text{integrity}(P,L))
\]

D. Security Policy for the ECA rules

1) Security Policy for Action Definitions: As presented in Section II-E, insecure information flows may occur due to action definitions relying on sensitive or untrusted conditions. Besides actions may perform write down or unsafe write. In order to control such information flows, we present security properties for the conditions, namely the Action Condition Security, and for the operations, namely the Action Effect Security. Let \( \text{do}(\alpha, Lc, Li) \text{ causes } op_1 \ldots op_k \text{ if } q_1(X_1), \ldots, q_n(X_n) \) be a multilevel action.

Action Condition Security: The execution of an action by user \( s \) at level \( Lc \) satisfies the Action Condition Security if for each condition \( q_i(X_i) \), the Access Law is satisfied w.r.t. \( q_i(X_i) \) and \( Lc \). This requirement enforces that for any action \( \alpha \) classified at level \( L \), if \( \alpha \) is executed in both states \( s_i \) and \( s'_i \), then \( s_i \equiv_L s'_i \Rightarrow s_{i+1} \equiv_L s'_{i+1} \).

Action Effect Security: The execution of an action by user \( s \) at level \( Lc \) satisfies the Action Effect Security if it satisfies (1) the User Privilege policy w.r.t. \( (s, Lc) \), (2) the Modification Law w.r.t. \( (p, Lc) \) if one \( op_1 \) operation is an insert of predicate \( p \), (3) the Access Law w.r.t. \( (p, Lc) \) if one \( op_1 \) operation is a select of predicate \( p \) and (4) both the Modification and Access Law w.r.t. \( (p, Lc) \) if one \( op_1 \) operation is a delete of predicate \( p \). This requirement enforces that for any action \( \alpha \) executed in state \( s_i \), if \( \alpha \) is classified at level \( Lc \), then for any level \( L \) such that \( L \nsubseteq Lc \), we have \( s_i \equiv_L s'_i \Rightarrow s_{i+1} \equiv_L s'_{i+1} \).

Action Integrity Security: The execution of an action by user \( s \) at the integrity level \( Li \) satisfies Action Integrity Security if it satisfies both the Integrity Evaluation law and the Integrity policy.

2) Security Policy for Event Definitions: As shown in Section II-E, events may create insecure information flows. Let \( \text{event}_\text{name}(X, Lc_e, Li_e) \) after \( \text{do}(\alpha, Lc, Li) \) if \( r_1(Z_1), \ldots, r_m(Z_m) \) be a multilevel event definition.

Event Condition Security: The Event Condition Security is satisfied if for each condition \( r_1(Z_1) \), the Access Law is satisfied w.r.t \( r_1(Z_1) \) and \( Lc_e \).

Event Action Security: The Event Action Security is satisfied if the security level \( Lc \) of the action \( \alpha \) is lower than or equal to the event security level \( Lc_e \).

Event Integrity Security: The event definition satisfies the Event Integrity Security if it satisfies the Integrity Evaluation policy: the integrity level \( Li_e \) of the event is lower than or equal to the lower bound of both the integrity levels of the event conditions and the integrity level \( Li \) of \( \alpha \).

3) Security Policy for Active Rule Definitions: Active rules have to satisfy several security properties to prevent insecure information flows. Indeed, actions launched by an event must not disclose any illegal information of the original action state. So let \( \text{event}_\text{name}(X, Lc_e, Li_e) \) initiates \( \text{do}(\alpha_1, Lc_1, Li_1) \ldots \text{do}(\alpha_j, Lc_j, Li_j) \) if \( t_1(Z_1), \ldots, t_p(Z_p) \) be an active rule definition.

Active Rule Condition Security: The active rule definition satisfies the Active Rule Condition Security if for each condition \( t_i(X_i) \), the Access Law is satisfied w.r.t \( t_i(X_i) \) and the greatest lower bound \( \text{glb}(Lc_1, \ldots, Lc_j) \).

Active Rule Action Security: The active rule definition satisfies the Active Rule Action Security if the security level of the event \( Lc_e \) is lower than or equal to each action security level \( Lc_1, \ldots, Lc_j \).

Active Rule Integrity Security: The Active Rule Integrity Security is satisfied if the integrity level \( Li_k \) of each action is lower than or equal to the lower bound of the integrity levels of both the active rule conditions and the integrity level \( Li \) of the event.

E. Declassification Policy

In order to avoid insecure information flows, the declassification policy has to satisfy several requirements. In this section, we define security properties for trust based and automatic declassification.

First, we define related notions. As the trustworthy aspect of a condition or an object depends on the security level of the object to declassify, we define the Safety w.r.t. a security level \( L \) and the Secure Declassification Operation Conditions properties as follows:

Safety Property: An object \( \text{obj} \) is said to be safe w.r.t. a security level \( L \) if the integrity level of \( \text{obj} \) being \( Li, L \subseteq Li \).

Secure Declassification Operation Conditions: When considering the declassification of \( P \), an action, event or rule satisfies the Secure Declassification Operation Conditions if its confidentiality level \( Lc \) satisfies \( Lc = \perp \) and its integrity level \( Li \) satisfies \( \text{classification}_\text{level}(P) \subseteq Li \).

Let \( \text{do}(\text{declassify}(P, L_{\text{dest}}), Lc, Li) \) be a query executed by \( U \). It enforces the Trust Based Declassification Control if \( P \) satisfies the safety property and \( \text{is}_\text{permitted}(U, \text{declassify}(P, L_{\text{dest}})) \). On the other hand, let this declassification be triggered by \( \text{declassify}_\text{event}(P, L_{\text{dest}}, Lc', L'_{\text{c}}) \). This declassification
enforces the Automatic Declassification Control if $P$ satisfies the safety property and the triggered event satisfies the Secure Declassification Operation Conditions. Based on these conditions and the confidentiality and integrity policies, the declassifications satisfy the confidentiality and integrity properties defined in Section III-B.

F. Proof of security enforcement

We can prove that the different security requirements defined in this section provide sufficient conditions to enforce the confidentiality and integrity properties defined in Section III. This proof has been done using the B method. For this purpose, we first need to specify the information system using B abstract machines. The proof is rather long and fastidious since it generates a large number of proof obligations. So, due to space limitation, we do not present the proof in this paper. Intuitively, we can derive various lemmas from the different enforcement requirements defined in this section. For example, the Action Condition Security requirement is a sufficient condition to enforce that for any action $α$ classified at level $L$, if $α$ is executed in both states $s_i$ and $s'_i$, then $s_i \models_L s'_i \Rightarrow s_{i+1} \models_L s'_{i+1}$. The final theorem is then proved by combining these different lemmas.

G. Time Management

As mentioned in [24], event based declassification operations may rely on time. For instance, the security policy of the United States requires that data have to be declassified to public ($\bot$) after 50 years.

In order to consider time based declassification rules, time has to comply with several constraints. First, as time is public, updates can be performed only at the public level. Second, time update has to be performed at the highest integrity level, else automatic temporal declassification may be blocked due to a violation of the Secure Declassification Operation Conditions. We can then associate declassify$_E$ event with the action clock which models time evolution. For example, let us consider the automatic downgrading requirement: the total war cost is automatically downgraded at the public level after a given delay when defeat occurs. This requirement is specified by the following declassify$_E$ event:

\[
\text{declassify}_E (\text{warCost}(C),P), \text{after do}(\text{clock}) \text{ if } \text{warCost}(C) \land \text{defeat} \land \text{end of war}(\text{Date}) \land \text{current date} = \text{Date} + \text{delay} \]

In this event definition we assume that predicate end of war$(\text{Date})$ means that end of war occurred at Date and that current date gives the current date.

H. ECA rules and declassification scenarios

In the previous sections, we defined our flow control and declassification policies. In this section, we present declassification scenarios which rely on these policies.

1) Enforcement of the security policy: When considering the confidentiality policy enforcement we first refine the security conditions for each action, event and rules. Second, for each action, event and rule, integrity constraints are added. Consider for instance an action which may triggers the event$(\epsilon,L_c,L_i)$ event. In order to satisfy the flow control policy, we thus say:

\[
\text{do}(\text{insert},O,L,L_i) \text{ causes Insert}(O) \text{ if } \text{classification}_E(\text{level}(O,L_c) \land \text{integrity}(O,L_s) \land L_s \subseteq L_i, \neg \text{classification}_E(\text{metadata}(O)))
\]

\[
\text{do}(\text{delete},O,L,L_i) \text{ causes Delete}(O) \text{ if } \text{classification}_E(\text{level}(O,L_c) \land \text{integrity}(O,L_s) \land L_s \subseteq L_i, \neg \text{classification}_E(\text{metadata}(O)))
\]

\[
\text{event}(\epsilon,L_c,L_i) \text{ after do}(\alpha,L_c,L_i) \text{ if } C \land L_c \subseteq L_i \land \text{classification}_E(\text{level}(C,L_c) \land L_s \subseteq L_i, \neg \text{classification}_E(\text{metadata}(O)))
\]

2) ECA rules for declassification actions: We first assume that the action declassify is only called by the active rules specified below. Indeed, as theses rules enforce the security policies, bypassing them would possibly leads to insecure declassification actions. Then, in order to enforce the Trust based Declassification action, we assume that users perform the request do$(U,\text{insert},O,L,L_i)$ where classification$_E$(metadata$(O)$). To enforce the Event based Declassification Action, we assume the associated actions trigger the event declassify$_E$(event$(U,O_c,L,L_i)$).

\[
\text{do}(\text{declassify}(O,L_{dest}),L,L_i) \text{ causes }
\text{Delete}(\text{classification}(O,L_{dest})) \text{ if } \text{classification}_E(\text{level}(O,L_c) \land \text{integrity}(O,L_s) \land L_s \subseteq L_i, \neg \text{classification}_E(\text{metadata}(O)))
\]

\[
\text{event}(\epsilon,L_{dest},L_i) \text{ after do}(\alpha,L_{dest},L_i) \text{ if } C \land L_{dest} \subseteq L_i \land \text{classification}_E(\text{level}(C,L_{dest}) \land L_s \subseteq L_i, \neg \text{classification}_E(\text{metadata}(O)))
\]

3) Modeling trust based declassification standards: Trust based declassification relies on the predicate is permitted$(U,\text{declassify}(O,L))$. In order to perform our trust based declassification rules, we need to refine is permitted. Consider a role based information system for which entities empowered in the role declassifier may declassified any contents. Such a role is defined as empowered$(U,\text{declassify}) \Rightarrow \text{is permitted}(U,\text{declassify}(O,L))$. 

V. Related Work

When considering the works presented in this paper, the first related domain of research is the database security area, as a concrete enforcement of our proposals may be the dynamic (or active) databases. In [3], Bertino et al. presented an overview of the database security challenges. They considered several issues like modeling of access control policy (System R, RBAC), management of multilevel databases and access control for recent database models (object oriented, XML). In [9], Cuppens et al. focused on multilevel databases. Based on observations about the polyninstantiation and its limitations, they proposed a model for multilevel databases with an explicit definition of cover stories. In [24], Sicherman et al. focused on modeling uncontrolled information leakage due to responses to queries. They showed that if attackers know the database policy, they may infer additional information, for instance in the case of a refusal based policy. Based on this observation, they defined a refusal based policy and proved its safety. In [6], Biskup et al. formally defined security properties for databases and proposed a model to enforce the confidentiality, which extends [23]. They studied the impact of known or unknown policies. They also evaluate the consequences of the databases security responses (lying or refusal). According to these studies, we can see that several works have been done to manage secret data. However, none of them simultaneously consider active rules or integrity properties. They are not able to address the declassification requirements stated in Section [11].

A second related domain of research is the modeling of declassification policies. In a recent study [21], Andrei Sabelfeld and David Sands summed up the different works made in this area and the issues to consider. They defined four basic dimensions a declassification system has to model: who? what? where? and when?. What? refers to modeling information to be declassified [13], [19]. The Abstract Non Interference model [18] proposed by Giacobazzi and Mastroeni weakens the notion of Non Interference with abstract variables. The Intransitive Non Interference model [19] aims at allowing information flows denied by the Non Interference policy. Who? refers to modeling the entity targeted by the declassification process [1]. Where? refers to the location of the declassification and When? refers to conditions constraining or triggering the declassification.

Firstly, the aforementioned works and those referenced in [21] do not explicitly control integrity flows. The only related dimension is what? but previous work only consider the downgraded object integrity [17]. Besides, when considering existing works, two scenarios to manage integrity levels are considered. On a first hand, strict integrity policies assume that

\[∀(U, P, L), (¬is_permitted(U, declassify_integrity(P, L))].\]

Thus, any write-up integrity flow is denied. On the other hand, we may refine existing Low-Watermark [4] based policies by requiring a two steps process which separately consider the integrity declassification and the object modification when is_permitted(U, declassify_integrity(P, L)). Our integrity declassification policy is also a two steps process which separately considers the integrity declassification and the object modification. However, it differs from existing Low-Watermark policies in several points. First, we do not specify a generic dynamic policy which allows any user to modify data. As stated in Section [11], this would allow low level users to prevent some data to be declassified. Second, we allow people to declassify the integrity level of an object without interfering with the data itself. This allows us to specify a more expressive policy which distinguishes declassification rights from access rights.

Secondly, existing declassification studies focus on (low level) programming languages. The declassification properties their are able to express is most of the time oriented on a specific dimension and lacks of a global definition. For example, the delimited release model [20] only assures that declassification flows occur in declassification requests. Besides, their proposal is unable to evaluate the robustness of a command when \(∃(i), ((M_1, e_i) ≠ (M_2, e_i)).\) When considering contextual conditions [20], [16], our work also relies on specified declassification points for events (the call to the declassification action). However, the evaluations of the information system security as the modeling of such “declassification points” widely differ. We also consider security definitions and constraints such as the trustworthiness of the declassification request.

Finally, existing declassification studies focus on (low level) programming languages. This prevents them from defining a concrete and expressive declassification policy which simultaneously considers the notions of user (who?), object (what?) action and condition (where? and when?). This indeed requires a higher view of the system.

VI. Conclusion

In this article, we show how to securely manage a complete set of declassification requirements in information systems, combining Trust based and Automatic declassification. We then present an information flow security model to manage declassification and control integrity policy of information to declassify. This model which combines confidentiality and integrity properties, is based on the concept of traces. Besides, we define security requirements to enforce this model in information systems with declassification and prove that these requirements are sufficient to assure our information
flow control model. Compared with previous works, our approach provides means to define expressive application dependent declassification policies using ECA rules and shows how such policies may be securely enforced. Previous language-based security approaches did not provide means to express such declassification policies.

In future work, we shall investigate the problem of declassification policy consistency. How to administrate a multilevel security policy that includes declassification requirements is another issue that requires further work. We are also investigating the enforcement of our security policy in real environments related to the ECA rules such as the Oracle database management system.

References


