On the Notion of Redundancy in Access Control Policies[*]

Marco Guarnieri [†]
Institute of Information Security, ETH Zürich, Switzerland
marco.guarnieri@inf.ethz.ch

Mario Arrigoni Neri
Università degli Studi di Bergamo, Italy
mario.arrigonineri@unibg.it

Eros Magri [†]
Comelit R & D,
Comelit Group S.p.A, Italy
eros.magri@comelit.it

Simone Mutti
Università degli Studi di Bergamo, Italy
simone.mutti@unibg.it

ABSTRACT

The evolution of information systems sees an increasing need of flexible and sophisticated approaches for the automated detection of anomalies in security policies. One of these anomalies is redundancy, which may increase the total cost of management of the policies and may reduce the performance of access control mechanisms and of other anomaly detection techniques.

We consider three approaches that can remove redundancy from access control policies, progressively reducing the number of authorizations in the policy itself. We show that several problems associated with redundancy are NP-hard. We propose exact solutions to two of these problems, namely the Minimum Policy Problem, which consists in computing the minimum policy that represents the behaviour of the system, and the Minimum Irreducible Policy Problem, consisting in computing the redundancy-free version of a policy with the smallest number of authorizations. Furthermore we propose heuristic solutions to those problems. We also present a comparison between the exact and heuristics solutions based on experiments that use policies derived from bibliographical databases.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Algorithms, Security, Theory

[*]This work was partially supported by the EC within the 7FP, under grant agreement 257129 “PoSecCo” and 256980 “NESSoS”, by the Italian Ministry of Research within the PRIN projects “PEPPER”, “GATECOM” and “GenData 2020”.

[†]This work was partially done while the author was with Dipartimento di Ingegneria, Università degli Studi di Bergamo, Italy.

Keywords

Redundancy, Access Control, Minimization

1 Introduction

Access control policies used in real systems are often unnecessarily large due to redundancy. Since the size of the policy is one of the main factors that determine the cost of managing the security configuration of a system, minimizing the size of a policy can ease the management of the policy itself and can reduce the cost of the management process. Furthermore the size of a policy influences the performance of the access control system, and thus minimizing the size of the policy can improve the access control system performance [11, 16].

Although *redundancy* seems a natural and quite simple concept, providing a formal definition of it in case of access control policies is not trivial for several reasons. One of the reasons is that we may have to deal with conflicts between authorizations that may influence the result of our redundancy detection process. Another reason is that at a certain point during our redundancy detection process we may consider as redundant a subset of the authorizations in the policy, and the choice of the authorizations to effectively tag as redundant may influence the redundancy of the others. In the following, we are going to survey several definitions of redundancy that were presented in the literature. Al-Shaer et al. in [1–3] define a rule r as *redundant* iff there are other rules that produce the same actions as r, such that the removal of r does not affect the security policy. A similar definition can be found in the work of Kolovski et al. [10], which defines a policy element as *redundant* if its removal does not change the final behavior of the policy. Also Liu et al. [11] and Yuan et al. [16] agree with the definition of redundancy given above by saying that a rule is redundant iff its removal does not influence at all the behavior of the firewall. All these authors give a very similar definition of *redundancy* (which we call *basic definition* in the following) and we can consider all the definitions as equivalent to saying that an authorization (or a rule) is *redundant* iff it does not affect the behavior of the access control mechanisms (i.e., its presence or absence in a policy passes unnoticed). The main problem of this definition is that, although it is very simple and intuitive, it often lacks in precision and flexibility: the definition of behavior of a policy is usually not specified in a formal way and it is only defined in terms of a specific and concrete access control system. Hu et al. [9] try to overcome this problem by providing a more formal definition, saying that a rule r is redundant in case the *authorization space* (which is a collection of access requests to which a policy element is applicable) derived from the policy that contains r is the same as the one derived from the policy without r.

Although the definition given above seems clear, it presents subtle side-effects that we show with a simple example. Let P be a policy composed by the authorizations a_1, a_2, a_3 where a_1 produces exactly the same effect as the combination of authorizations a_2 and a_3.

A first problem is that the *basic definition* of *redundancy* is not invariant w.r.t. the decisions of our redundancy-removal process. This fact means that a rule may be considered redundant at a certain point in time during the redundancy-removal process, and lose this property at a later time. For instance, if we consider the policy P, the redundant authorizations are a_1, a_2, a_3 (i.e., the complete policy), but if we remove authorization a_3, then a_1 is not redundant anymore. This means that we cannot safely remove sets of redundant authorizations and the definition cannot be applied looking only at the starting policy state. The definition of *redundancy* should take into account all the decisions taken during the redundancy-removal process.

Furthermore, by iteratively removing authorizations that satisfy the *basic definition*, we usually do not obtain a unique solution. Rather, we can obtain several equivalent policies with different number of authorizations. For instance, in our example both policies $\{a_1\}$ and $\{a_2, a_3\}$ are equivalent and redundancy-free, but they have different size. The fact that the *basic definition* of *redundancy* does not take into account the number of authorizations in the final policy can be surprising, since the main motivation behind the development of redundancy detection techniques is the improvement of access control mechanisms' performance, which primarily depends on the size of the access control policies. This aspect of the redundancy problem is important, because an effective redundancy-removal process should always aim at computing the minimum redundancy-free version of the given policy, not limiting the goal to the identification of one of the redundancy-free versions.

An underlying assumption in all the previous definitions is that the only way on which we can act on a policy is by removing redundant authorizations. This assumption introduces some limitations in the search for performance improvements. For instance, if P is a policy with 6 authorizations, there may exist another policy P' that is equivalent to P but it contains only 4 authorizations that were not in P. In this case, we should prefer P' over P because it can lead to an improvement in system performance. We consider as another aspect of the *redundancy* problem the computation of the policy that models the behavior of the system with the minimum number of authorizations.

It is then obvious that the *basic definition* of *redundancy* is not enough for handling this issue effectively. There is the need of one or more definitions for several aspects of the redundancy problem that take into account the size of the resulting policy, and the dependency between actions performed at different steps of the redundancy-removal process.

In this work we present a formalization of the redundancy problem in access control policies that considers three different ways in which a security administrator can act on a policy containing redundancy. In the first approach, the administrator can compute an equivalent policy that does not contain redundancy anymore, i.e., she computes an *irreducible policy*. However, given a certain policy there usually are several irreducible versions of the same policy. Given the fact that the number of authorizations of the policy is one of the major factors that influence the management cost of the policy itself, in the second approach the security administrator identifies among the *irreducible policies* obtained by the original policy one with the minimum number of authorizations, which is the *minimum irreducible policy*. With the third approach, the security administrator may be interested in computing the representation of the access control system with the minimum number of authorizations, i.e., the *minimum policy*.

Main contributions: The main contributions of our work are: (a) we propose a definition of the redundancy removal problem in access control policies; (b) we study in detail two new problems related with redundancy; (c) we provide two exact solutions and two heuristic solutions for these problems; (d) we provide an evaluation of the various approaches based on data extracted from bibliographical databases.

Structure of the paper: Section 2 describes the underlying model used in our approach. In Section 3 we study in detail the problem of removing redundancy from security policies. Section 4 presents exact solutions and heuristic algorithms for solving two different redundancy-removal problems; we present also a detailed performance analysis. In Section 5 we illustrate an analysis of the state of the art with respect to redundancy detection. Finally, Section 6 draws our conclusions.

2 Model

In enterprise scenarios, policy definition and management systems are needed that provide a high level of flexibility, to correctly represent the large number and variety of security requirements. In order to achieve this goal, we have defined an access control model containing the following entities:

- **Principals**: represent users and groups of principals. Each *Principal* may contain one or more other *Principals* and this fact is represented by the function *contains*:$Principal \rightarrow 2^{Principal}$. The function *contains+*: $Principal \rightarrow 2^{Principal}$ is the transitive closure of *contains*.
- **Actions**: represent the actions that users can execute. Each *Action* may be composed by one or more *Actions* and this fact is represented by the function *composed*:$Action \rightarrow 2^{Action}$. The function *composed+*: $Action \rightarrow 2^{Action}$ is the transitive closure of *composed*.
- **Resources**: represent the resources on which users can act. Each *Resource* may contain one or more *Resources* and this fact is represented by the function *containsResources*:$Resource \rightarrow 2^{Resource}$. The function *containsResources+*:$Resource \rightarrow 2^{Resource}$ is the transitive closure of *containsResources*.

An instance $\mathcal{M}$ of our model is a list of *Principals*, *Actions* and *Resources* with the associated functions[1]. $\preceq_P$, $\preceq_A$ and $\preceq_R$ are partial orders defined over *Principals*, *Actions* and *Resources* respectively. For instance, given two principals p_1 and p_2 we say that $p_1 \preceq_P p_2$ iff $p_1 \in contains+(p_2) \cup \{p_2\}$. The definition of $\preceq_A$ and $\preceq_R$ can be obtained in a similar way (the fact that $\preceq_P$, $\preceq_A$ and $\preceq_R$ are partial orders is

[1]The hierarchies defined in $\mathcal{M}$ must be acyclic.

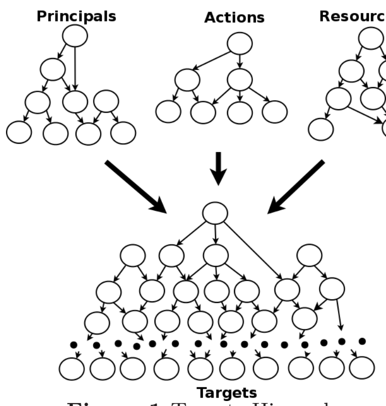


Figure 1 Targets Hierarchy

enforced by the fact that the hierarchies over *Principals*, *Actions* and *Resources* are acyclic). We say that an element of one of the hierarchies in $\mathcal{M}$ is *primitive* iff it is a leaf of the hierarchy (e.g., a principal p is a primitive element iff $contains(p) = \emptyset$).

We assume that the assignment of permissions to users can be derived from the system, e.g. in Role-Based Access Control (RBAC) [14] the user-permission assignment matrix can be directly computed from the user-role assignment and the role-permission assignment matrices.

The basic element about which we can express access control decisions is called a *Target*, and it is defined in the following way:

Definition 1. **Target:** a target consists of a *Principal* p, an *Action* a and a *Resource* r. We represent a target as a triple $< p, a, r >$. We say that a target $< p, a, r >$ is *primitive* iff p, a and r are primitive elements.

Given two targets $t_1 =< p_1, a_1, r_1 >$ and $t_2 =< p_2, a_2, r_2 >$, we say that t_1 implies t_2, denoted by $t_1 \preceq_T t_2$, iff $p_1 \preceq_P p_2 \wedge a_1 \preceq_A a_2 \wedge r_1 \preceq_R r_2$. The partial order $\preceq_T$ defines the *Target Hierarchy* $TH_{\mathcal{M}}$, shown in Figure 1. Primitive targets are the leafs of the hierarchy.

Security administrators can define access rights on the targets by means of authorizations, which are defined in the following way:

Definition 2. **Authorization**: An authorization consists of a triple composed by a set of *Principals* P, a set of *Actions* A and a set of *Resources* R. Each authorization has a sign s that can be $+$ or $-$. It is used in order to state, respectively, whether an authorization is *positive* (i.e., it grants the permission to do something) or *negative* (i.e., it denies the permission to do something). We graphically represent an authorization in the following way: $< s, P, A, R >$.

An authorization $auth =< s, P, A, R >$ defined over the model $\mathcal{M}$ is associated in an unique way to a set of *Targets* $T_{auth} = P \times A \times R$ on which the authorization acts. Without loss of generality, we can express access control decisions only in terms of primitive targets (and thus we consider only primitive targets when we compute the set T_{auth}). Given an authorization, the following functions can be used in order to obtain the elements contained in it:

- *principals*:*Authorization*$\rightarrow 2^{Principal}$ retrieves the set of *Principals* involved in the authorization,

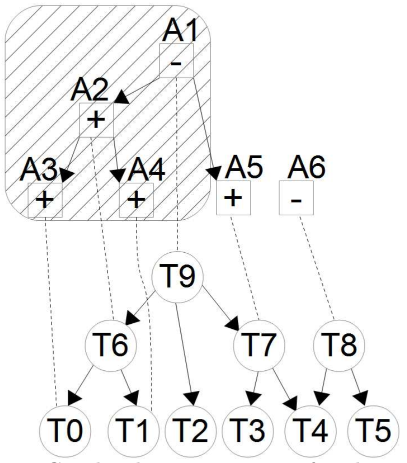


Figure 2 Graphical representation of authorizations

- *actions*:*Authorization*$\rightarrow 2^{Action}$ retrieves the set of *Actions* involved in the authorization,
- *resources*:*Authorization*$\rightarrow 2^{Resource}$ retrieves the set of *Resources* involved in the authorization,
- *sign*:*Authorization*$\rightarrow \{+, -\}$ retrieves the *sign* of the authorization.

Given a set of authorizations Δ and an authorization $a \in \Delta$, the *region* defined by a over Δ is $R_a^{\Delta} = \{a' \in \Delta | T_a \cap T_{a'} \neq \emptyset\}$. The region of an authorization a contains all the authorizations that may interact with a. We can define a hierarchy over authorizations in the following way: given two authorizations $auth_1$ and $auth_2$, we say that $auth_1$ is dominated by $auth_2$, i.e., $auth_1 \preceq auth_2$, iff $T_{auth_1} \subseteq T_{auth_2}$.

Example 1. Figure 2 shows a graphical representation of a set of authorizations applied over a set of targets. This example will be used in the next sections as running example. The graph is composed by two different components (a) the authorizations, represented by squares labeled with a sign, and (b) the targets, represented by circles (this set of targets represent a part of the target hierarchy shown in Figure 1). Edges between authorizations represent the $\preceq$ ordering relation, whereas edges between targets represent the $\preceq_T$ ordering relation. In this representation, given an authorization $auth$ the set T_{auth} is defined by all the nodes in the target hierarchy reachable from the node representing $auth$. For instance, if we consider the authorization $A2$ then T_{A2} is the set $\{T0, T1, T6\}$. The region defined by the authorization $A2$ is $\{A1, A2, A3, A4\}$.

Let $auth$ be an authorization. $auth$ specifies an access control decision on the targets on which it can act (represented by the sign $sign(auth)$), and we represent this fact by saying that $auth$ assigns a label $sign(auth)$ to the targets in T_{auth}. We represent the basic access control decision by means of the concept of *Privilege*, which represents a *Target* labeled with a sign s, defined in the following way:

Definition 3. **Privilege**: A privilege consists of a target $t =< p, a, r >$ and a sign $s \in \{+, -\}$ (inherited from an authorization). We graphically represent a privilege in the following way: $< s, p, a, r >$. If the sign s is $+$ then the privilege represents the fact that the *Principal* p is allowed to do the *Action* a on the *Resource* r. On the contrary, if the sign s is $-$ then the privilege represents the fact that the

Principal p is not allowed to do the *Action* a on the *Resource* r. We say that a privilege $< s, p, a, r >$ is *primitive*, i.e., it does not imply other privileges, iff the target $< p, a, r >$ is primitive.

Given a privilege the following functions can be used in order to obtain the elements of the model contained in it:

- *principal*:*Privilege*→*Principal* retrieves the *Principal* involved in the privilege,
- *action*:*Privilege*→*Action* retrieves the *Action* involved in the privilege,
- *resource*:*Privilege*→*Resource* retrieves the *Resource* involved in the privilege,
- *sign*:*Privilege*→ $\{+, -\}$ retrieves the *sign* of the privilege.

Each authorization *auth*, thus, grants a set of privileges (i.e., a set of labeled targets). More formally, we say that each authorization *auth* associates a sign s to the targets in the set T_{auth}. The procedure shown in Algorithm 1 can be used to compute the set of privileges associated with an authorization. We denote the set of privileges associated with the authorization *auth* as *privileges(auth)*.

Algorithm 1: *privileges* procedure for authorizations

```
Input  : Authorization $auth$
Output: $Privileges$
begin
    $Privileges = \emptyset$;
    for $p \in principals(auth)$, $a \in actions(auth)$,
    $r \in resources(auth)$ do
        for $p' \in contains+(p) \bigcup \{p\}$,
        $a' \in composed+(a) \bigcup \{a\}$,
        $r' \in containsResources(r) \bigcup \{r\}$ do
            if $contains(p') = \emptyset \wedge composed(a') =$
            $\emptyset \wedge containsResources(r') = \emptyset$ then
                $Privileges = Privileges \bigcup \{<$
                $sign(auth), p', a', r' >\}$;
```

Due to the fact that authorizations and privileges have a sign, we may have conflicts between different authorizations and privileges in the same policy P. Several approaches have been proposed in the literature for the solution of conflicts between authorizations [6, 12, 15]. Our approach is not tied to any particular conflict resolution strategy, indeed it can be used with any conflict resolution strategy that satisfies certain requirements, explained in detail in Section 2.1. Let $\mathcal{M}$ be a model, let $P_{\mathcal{M}}$, $A_{\mathcal{M}}$ and $R_{\mathcal{M}}$ be the set of principals, actions and resources associated with the model, and let Δ be the set of all possible authorizations that can be defined over $\mathcal{M}$. We represent a conflict resolution strategy as a function $\psi : P_{\mathcal{M}} \times A_{\mathcal{M}} \times R_{\mathcal{M}} \times \mathbb{P}(\Delta) \rightarrow \Delta \cup \{\perp\}$ that takes as input a target $< p, a, r >$ and a set of authorizations A and returns the authorization in A that determines the privilege granted by the set A w.r.t. the $< p, a, r >$ (if no authorization in Δ can be applied to $< p, a, r >$ then ψ returns $\perp$).

Let Δ be a set of authorizations, we denote with *privileges*(Δ, ψ) the set of privileges granted by Δ w.r.t. the conflict resolution strategy ψ. In this case *privileges*(Δ, ψ) can be computed by using the procedure shown in Algorithm 2 that returns the set of labeled targets associated with Δ w.r.t. the strategy ψ. Given a set of authorizations Δ, a conflict resolution strategy ψ and an authorization *auth*, we say that Δ dominates *auth* w.r.t. ψ, denoted by $auth \preceq \Delta$, iff *privileges*$(auth) \subseteq$ *privileges*(Δ, ψ).

Algorithm 2: *privileges* procedure for sets of authorizations

```
Input  : Set of authorizations $\Delta$, Conflict resolution strategy $\psi$
Output: $Privileges$
begin
    $Privileges = \emptyset$;
    for $auth \in \Delta$ do
        for $p \in principals(auth)$, $a \in actions(auth)$,
        $r \in resources(auth)$ do
            for $p' \in contains+(p) \bigcup \{p\}$,
            $a' \in composed+(a) \bigcup \{a\}$,
            $r' \in containsResources(r) \bigcup \{r\}$ do
                if $contains(p') = \emptyset \wedge composed(a') =$
                $\emptyset \wedge containsResources(r') = \emptyset$ then
                    $a = \psi(p', a', r', \Delta)$;
                    $Privileges = Privileges \bigcup \{<$
                    $sign(a), p', a', r' >\}$;
```

Definition 4. **Policy**: A policy consists of a set of authorizations Δ and a conflict resolution strategy ψ. We represent a policy as a pair $< \Delta, \psi >$.

Let $P =< \Delta, \psi >$ be a policy, we denote with $P' \subset P$ a policy $P' =< \Delta', \psi' >$ such that $\Delta' \subset \Delta$ and $\psi' = \psi$.

Each policy is identified by its behaviour, which is defined in the following way:

Definition 5. **Behaviour of a Policy**: the behaviour of a policy P is the set of privileges granted, directly or indirectly, by the policy.

The behaviour of a policy P can be computed using the function *privileges* presented in Algorithm 2, thus we can check whether two different policies are equivalent by checking whether they enable the same set of privileges. The behaviour of the policy models how the real access control system behaves. Given the fact that the same behaviour can be modeled by means of several different policies, the equivalence relation between policies is defined in the following way:

Definition 6. **Equivalence of Policies**: Two policies $P =< \Delta, \psi >$ and $P' =< \Delta', \psi' >$ are equivalent, $P \equiv P'$, iff they have the same behaviour, i.e., iff *privileges*$(\Delta, \psi) =$ *privileges*(Δ', ψ').

Let P be a policy defined over the model $\mathcal{M}$. There exists always a policy P' expressed only in terms of primitive elements of $\mathcal{M}$ which is equivalent to P and such that $|P| = |P'|$. In order to obtain P' we can proceed in the following way: we define an authorization $< s, Pr', A', R' > \in P'$ for each authorization $< s, Pr, A, R > \in P$ where Pr', A' and R' contains only the primitive elements that can be derived from the elements in Pr, A, and R respectively. The fact that $P \equiv P'$ follows trivially from the definition of equivalence and from the *privilege* procedure. In the following we can thus consider, without loss of generality, policies expressed only in terms of primitive elements.

2.1 Conflict resolution strategies

Let $\mathcal{M}$ be a model, let $P_{\mathcal{M}}$, $A_{\mathcal{M}}$ and $R_{\mathcal{M}}$ be the set of principals, actions and resources associated with the model and let Δ be the set of all possible authorizations that can be defined over $\mathcal{M}$.

A conflict resolution strategy is a function $\psi : P_{\mathcal{M}} \times A_{\mathcal{M}} \times R_{\mathcal{M}} \times \mathbb{P}(\Delta) \rightarrow \Delta \cup \{\bot\}$. ψ takes as input a set of authorizations Δ and a target $< p, a, r >$, and it returns as output the authorization $auth \in \Delta$ that is applied over $< p, a, r >$ according to the strategy (if no authorization in Δ can be applied to $< p, a, r >$ then ψ returns $\bot$).

A conflict resolution strategy ψ may satisfy one or more of the following properties:

- **Polynomiality:** we say that ψ is polynomial (in the size of Δ and of the target hierarchy) iff there is an algorithm that implements ψ which is in **P**,
- **Completeness:** we say that ψ is complete iff for any possible set of authorizations Δ and for any target $t =< p, a, r >$ if $\exists auth \in \Delta :< p, a, r >\in T_{auth}$ then $\psi(p, a, r, \Delta) \neq \bot$,
- **Monotonicity:** we say that ψ is *monotone* iff for any possible set of authorizations Δ and for any target $t =< p, a, r >$ then $\psi(p, a, r, \Delta) = \psi(p, a, r, \Delta \cup \{auth\})$ holds for any authorization $auth$ such that $< p, a, r >\notin T_{auth}$. A monotone conflict resolution strategy is one that produces a result that depends only on the authorizations in Δ that are related with $< p, a, r >$ (i.e., adding unrelevant authorizations do not change the outcome of the strategy).

We say that ψ is a *valid* conflict resolution strategy for our framework, iff ψ satisfies the three properties above.

For instance, we can represent the *Denial takes precedence* strategy in the following way:

$$\psi_{DTP}(p, a, r, \Delta) = \begin{cases} a_i & \textit{if } < +, p, a, r >\in \textit{privileges}(a_i) \wedge \\ & \nexists a_j \in \Delta \setminus \{a_i\} : \\ & < -, p, a, r >\in \textit{privileges}(a_j) \\ a_i & \textit{if } < -, p, a, r >\in \textit{privileges}(a_i) \\ \bot & \textit{otherwise} \end{cases}$$

It is easy to see that the *Denial takes precedence* strategy satisfies all the requirements stated above.

Example 2. If we consider only the authorizations *A5* and *A6* of Figure 2, we can define the sets (a) $T_{A5} = \{T3, T4, T7\}$, and (b) $T_{A6} = \{T4, T5, T8\}$. We can notice that there is an intersection between T_{A5} and T_{A6}, and the two authorizations have a different sign. The application of the strategy *Denial takes precedence* means that for the target *T4* (i.e., the intersection) will be applied the authorization *A6*, the negative one.

The *Most Specific Wins* strategy can be represented by the following function ψ_{MSW}:

$$\psi_{MSW}(p, a, r, \Delta) = \begin{cases} a_i & \textit{if } < p, a, r >\in T_{a_i} \wedge \\ & \nexists a_j \in \Delta \setminus \{a_i\} : \\ & (< p, a, r >\in T_{a_j} \wedge a_j \preceq a_i) \\ \bot & \textit{if } \exists a_i, a_j \in \Delta :< p, a, r >\in T_{a_i} \\ & \wedge < p, a, r >\in T_{a_j} \wedge \\ & a_i \npreceq a_j \wedge a_j \npreceq a_i \\ \bot & \textit{otherwise} \end{cases}$$

It is easy to see that ψ_{MSW} is not complete because it may return $\bot$ also in case there are authorizations in Δ that can be applied to the target $< p, a, r >$ but these authorizations are not comparable.

Example 3. If we consider only the authorizations *A2* and *A3* of Figure 2, we can define the sets (a) $T_{A2} = \{T0, T1, T6\}$, and (b) $T_{A3} = \{T0\}$. We can notice that $A3 \preceq A2$ because $T_{A3} \subseteq T_{A2}$. For instance, for the target *T0* we can apply two authorizations *A2* and *A3*. The *Most Specific Wins* strategy chooses to apply the authorization $A3$ because it is more specific than $A2$.

Although our framework is independent from a specific conflict resolution strategy, for concreteness in the the running example and in Section 4 we consider the conflict resolution strategy ψ that applies the *Most Specific Wins* and *Denial takes precedence* criteria. This strategy can be modeled in the following way:

$$\psi(p, a, r, \Delta) = \begin{cases} \psi_{MSW}(p, a, r, \Delta) & \textit{if } \psi_{MSW}(p, a, r, \Delta) \neq \bot \\ \psi_{DTP}(p, a, r, \Delta) & \textit{otherwise} \end{cases}$$

ψ satisfies all the requirements stated above.

3 Redundancy

Sometimes real policies contain redundancy [13], for several reasons caused by the evolution of security policies during time. Although the concept of *redundancy* is easy to understand, defining it formally is not trivial, especially, as in our case, when we consider conflicts.

A simple definition of *redundancy* may be the following: "Given a policy P, we can define an authorization $auth$ as *redundant* when it does not add anything to the behaviour of P.". The process of removing redundancy is called *redundancy-removal* process. The problem of this definition is that redundancy is not invariant, i.e., it depends on the sequences of decisions taken during the redundancy-removal process. Indeed, by solving a conflict between authorizations or by removing an authorization from the policy, we may change the set of authorizations that are redundant. As a consequence, the sequence of decisions taken during the redundancy-removal process in order to achieve a redundancy-free equivalent policy, may lead to significantly different results in terms of size of the final policy.

In the following we try to formalize the *redundancy* problem. The proof of all the theorems are given in the appendix.

We start by defining the concept of *redundancy condition*, which refines the definition given above.

Definition 7. **Redundancy Condition**: An authorization $auth \in P$ satisfies the *redundancy condition* w.r.t. the policy $P =< \Delta, \psi >$ iff $\forall < p, a, r >\in T_{auth}$:
$sign(\psi(p, a, r, R^{\Delta}_{auth} \setminus \{auth\})) = sign(\psi(p, a, r, R^{\Delta}_{auth})) \wedge$
$\psi(p, a, r, R^{\Delta}_{auth} \setminus \{auth\}) \neq \bot$.

In other words, $auth$ satisfies the redundancy condition w.r.t. P iff its presence or absence does not influence the access control decision for any possible target $< p, a, r >$ on which it can act.

Example 4. For instance, authorization $A2$ in Figure 2 satisfies the redundancy condition. The set T_{A2} contains the targets $T0$, $T1$ and $T6$ and for each of these targets the presence of $A2$ does not influence the behaviour of the policy. Hence, we can safely remove $A2$.

Given a policy P, we can remove one by one all the authorizations that satisfy the redundancy condition without changing the behaviour of the policy. However, by removing authorizations from P we may change the redundancy condition for other authorizations. We model this phenomenon with the concept of *sequence of reductions*.

Definition 8. **Sequence of Reductions (SoR):** let P_0 be a policy, and let $P_1, \ldots, P_n$ be a sequence of policies. The sequence $P_0, P_1, \ldots, P_n$ is a *sequence of reductions* iff it satisfies the following requirements:

- $P_n \subset P_{n-1} \subset \ldots \subset P_1 \subset P_0$,
- for each $i \in \{1, \ldots, n\}$, $P_{i+1} = P_i \setminus \{\gamma_i\}$ and γ_i is an authorization that satisfies the *redundancy condition* w.r.t. P_i,
- there are no authorizations in P_n that satisfy the redundancy condition w.r.t. P_n.

We say that P_0 is the *begin* of the sequence, and that P_n is the end of the sequence. We say that the reduction from P_i to P_{i+1} is a *step* in the sequence.

Due to the fact that the authorization removed at each step satisfies the redundancy condition, the behaviour of the policy does not change along the sequence of reduction, as stated in Theorem 1.

Theorem 1. Let $P_0, \ldots, P_n$ be a sequence of reductions. All the policies in the sequence have the same behaviour, i.e., they are equivalent.

An interesting property of *SoRs* is that once we reach a point P_i in a sequence $P_0, \ldots, P_n$ then all the authorizations that satisfy the redundancy condition at P_i satisfy the redundancy condition also at the following steps of the sequence, as stated in Theorem 2. This means that, although the choice of the authorization to remove may influence the final outcome, the order in which we remove these authorizations does not influence the result (i.e., what influences the size of the final policy is only $\Gamma = \{\gamma_0, \ldots, \gamma_{n-1}\}$ and not the order in the SoR in which we remove authorizations that satisfy the redundancy condition).

Theorem 2. Let $P_0, P_1, \ldots, P_n$ be a sequence of reductions. Let $auth \in P_0$ be an authorization such that there exists a value $j \in \{1, \ldots, n\}$ for which $\forall j' < j : auth \in P_{j'}$ and $auth$ satisfies the redundancy condition w.r.t. P_j. In this case $auth$ satisfies the redundancy condition w.r.t. all $P_i \cup \{auth\}$ where $j < i \leq n$.

Another important property of *SoRs* is stated in Theorem 3. The theorem says that whenever there are two equivalent policies P and P' such that P' is a subset of P, then there is always at least a sequence of reductions that starts from P and passes through P'. This means that we can reduce the problem of finding a redundancy-free version of a policy P to the one of finding an adequate *SoR* starting from P.

Theorem 3. Let P be a security policy and let $P' \subset P$ be another policy such that P and P' are equivalent. There is always at least one sequence of reductions of the form $P, \ldots, P', \ldots, P_n$.

We can now define the concept of redundancy-free policy, called *irreducible policy*, in the following way:

Definition 9. **Irreducible Policy (IP)**: A policy P is an irreducible version of the policy P' iff it does not contain any authorization that satisfies the redundancy condition w.r.t. P, $P \equiv P'$, and $P \subseteq P'$. This is equivalent to say that there is a sequence of reductions from P' to P, i.e., $P', \ldots, P$.

From Theorem 3 follows that in case we reach a policy P in a SoR such that there are no authorizations satisfying the redundancy condition w.r.t. P, then we can soundly say that an equivalent policy $P' \subset P$ does not exist. Hence given a policy P, we can compute an equivalent irreducible version P' in a simple, although sometimes inefficient, way. Initially let $P' = P$, then we iterate over all the authorizations in P', and we check for each authorization $a \in P'$ whether it satisfies the redundancy condition w.r.t. P' or not, if this is the case then we remove the authorization (i.e., $P' = P' \setminus \{a\}$). We iterate the above procedure until no more authorizations satisfy the redundancy condition. It is easy to see that this algorithm is polynomial w.r.t. the number of authorizations in P.

Checking whether a certain policy P is irreducible or not is the Irreducible Policy Problem (IPP) and it can be solved in **P-TIME**, as demonstrated by Theorem 4.

Definition 10. **Irreducible Policy Problem**: Given a policy P, checking whether P is irreducible is called Irreducible Policy Problem (IPP).

Theorem 4. The *IPP* is in **P**.

Example 5. Figure 3(a) shows an irreducible version of the policy in Figure 2. The new policy was obtained by removing the authorization $A2$ which satisfies the redundancy condition, as shown in Example 4.

Given a policy P, several different irreducible versions of it may exist. In order to improve the performance of access control mechanisms, a possible solution is to compute the irreducible version of P with the minimum number of authorizations. We call this policy a *Minimum Irreducible Policy*.

Definition 11. **Minimum Irreducible Policy (MIP)**: A policy P is a minimum irreducible version of the policy P' iff a policy P'' does not exist such that P'' is irreducible, P'' is equivalent to P', $|P''| < |P|$, $P \subseteq P'$ and $P'' \subseteq P'$. In an equivalent way, we can say that P is a minimum irreducible version of the policy P' iff P is the end of the longest SoR that can be computed starting from P'.

The problem of checking whether a certain policy P is a minimum irreducible policy with respect to the original policy P' is called Minimum Irreducible Policy Problem.

Definition 12. **Minimum Irreducible Policy Problem**: Given two policies P and P', checking whether P is a minimum irreducible policy with respect to P' is called Minimum Irreducible Policy Problem (MIPP).

We are more interested in the the problem of finding a minimum irreducible version of a given policy P (which is the search problem associated with the MIPP). Since the associated decision problem is **coNP-complete**, the search problem is **NP-hard**.

Theorem 5. The *MIPP* is **coNP-complete**.

Example 6. Although in Example 4, we have shown a redundancy-free version of the policy in Figure 2, the resulting policy (obtained by removing $A2$) was not the minimum irreducible one. Indeed a smaller irreducible policy can be computed by removing $A3$ and $A4$ instead of $A2$. This policy, shown in Figure 3(b), is the minimum irreducible one, since no smaller irreducible policies exist.

Given the fact that the behaviour of an access control system may be modeled by means of different equivalent policies, security administrators may be interested in computing the policy with the minimum number of authorizations that models the system, i.e., the *minimum policy.*

Definition 13. **Minimum Policy (MP)**: A policy P is said to be minimum iff an equivalent policy P' does not exist such that $|P'| < |P|$.

Given a policy P we can compute an irreducible policy P' equivalent to P by removing authorizations that satisfy the redundancy condition. However, in order to compute the minimum policy P'' we may have to define new authorizations that do not exist in P or remove authorizations in P with the only constraint that the resulting policy P'' must have the same behaviour as the original one. The problem of checking whether a policy is minimum or not can be defined in the following way:

Definition 14. **Minimum Policy Problem**: Given a policy P, checking whether P is minimum is called Minimum Policy Problem (MPP).

Theorem 6. The *MPP* is **coNP-complete**.

We are more interested in the the problem of computing a minimum version P' of a given policy P (which is the search problem associated with the MPP). Since the associated decision problem is **coNP-complete**, the search problem is **NP-hard**. It is worth pointing out that depending on the given policy P, the search problems associated with *MIPP* and *MPP* may do not have a unique solution, i.e., there may be several different *minimum irreducible* and *minimum* versions of the same policy P (all with the same size).

Example 7. Although the policy computed in Example 6, is the minimum irreducible policy, a smaller policy exists and it is shown in Figure 3(c)[2]. This policy is the minimum policy.

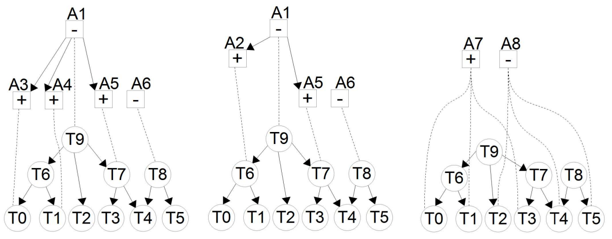
4 Implementation

In this section, we present techniques for solving the *MIPP* and *MPP* problems. We ignore the *IPP* because several algorithms were proposed in the literature to solve this problem [5,9–11,16] (and also because any solution for *MIPP* is a solution for *IPP*). In the following we show how *MIPP* and *MPP* can be mapped on the *Weighted SAT* problem. The *Weighted SAT* problem is an extension of the SAT problem, and it is defined as follows:

Definition 15. **Weighted SAT**: Given a set of variables U and a collection C of clauses over U, computing, in case it exists, the truth assignment $t : U \to \{0, 1\}$ which satisfies C and minimizes a certain cost function $\sum_{u_i \in U'} k_i * u_i$, where $U' \subseteq U$ and $k_i \in \Re$, is called the **Weighted Satisfiability Problem**.

[2]We assume that the parent node is equal to the union of its children.

Section 4.1 presents how the *MIPP* can be mapped to the *Weighted SAT* problem, whereas in Section 4.2 we present a heuristic technique for solving *MIPP*. In Section 4.3 we present how the *MPP* can be mapped to the *Weighted SAT* problem. In Section 4.4 we present an algorithm that uses a heuristic approach to solve *MPP*. Section 4.5 presents some experimental results.



(a) IP (b) MIP (c) MP

Figure 3 Redundancy-free policies

4.1 MIPP to Weighted SAT

We can map the *MIPP* to the *Weighted SAT* problem, and we can use efficient *SAT*-solvers in order to identify the minimum irreducible version of the input policy. Let $P =< \Delta, \psi >$ be the input policy, we want to produce a policy $P' =< \Delta', \psi >$ such that $\Delta' \subseteq \Delta$ and P' is a minimum irreducible version of P. We denote with G the set *privileges*(Δ, ψ).

We define a variable a_i for each authorization $a_i \in \Delta$. Due to the fact that the correspondence between variables and authorizations is clear, in the following we switch freely between the two notations (e.g., sometimes a_i may refer to an authorization and sometimes it may refer to the variable associated with that authorization). For simplicity's sake, we do not present formulae in CNF (this is not a problem because it is always possible to translate a formula to an equivalent one in CNF).

The cost function $min \sum_{a_i \in \Delta} a_i$ aims at minimizing the number of authorizations in the resulting policy (and thus it guarantees that the resulting policy is the minimal irreducible one).

We still have to handle conflicts and the conflict resolution strategy in our *Weighted SAT* instance. Our approach considers a general conflict resolution strategy ψ. Let $t =< p, a, r >$ be a target defined in our model, we denote with K_t the set containing all the authorizations that act on the target t (i.e., $K_t = \{a_i \in \Delta | t \in T_{a_i}\}$). Given a target $t =< p, a, r >$ and a privilege $p =< s, p, a, r >$ we define $C(p)$ in following way:

$$C(p) = \bigvee_{X \in \mathbb{P}(K_t) \wedge sign(\psi(p,a,r,X)) = sign(p)} \Big(\bigwedge_{a_i \in X} a_i \wedge \bigwedge_{a_i \in K_j \setminus X} \neg a_i \Big)$$

For each privilege $p_j \in G$ we add a constraint in the form $C(p_j)$, this constraint enumerates all the possible combinations of authorizations that effectively grant the privilege p_j. By enforcing these constraints we ensure that the behaviour of the resulting policy is equivalent to the behaviour of P. An authorization $a_i \in \Delta$ is in Δ' iff the variable associated with a_i is set to one in the result of the *Weighted SAT* problem. It is easy to see that the result of the *Weighted SAT* problem produces a *Minimum Irreducible Policy.*

4.2 Heuristic Algorithm for MIPP

As described above the MIPP is a **coNP-complete** problem. Hence, in this Section we propose a heuristic algorithm that allows to find, in an efficient way, an irreducible policy close to one of the exact solutions.

Algorithm 3: Heuristic Algorithm for MIPP

```
Input  : Policy $P$, Conflict Resolution Strategy $CRS$
Output: MIP $P$
begin
    $bool\ removed$;
    repeat
        $removed = false$;
        for $a \in P$ do
            $List\ region = computeRegion(a, P)$;
            if $isRemovable(a, region, CRS)$ then
                $P.removeAuthorization(a)$;
                $removed = true$;
    until $removed$;
```

Our approach, shown in Algorithm 3, is based on an iterative process. Let $P =< \Delta, \psi >$ be the initial policy. At each iteration, the algorithm iterates over the authorizations and for each authorization a computes its region R_a^{Δ} by means of the *computeRegion* procedure. Then the algorithm checks whether the authorization satisfies the redundancy condition w.r.t. P by means of the *isRemovable* procedure, which takes as input (a) the selected authorization, (b) the region, and (c) the resolution strategy. If the authorization a satisfies the redundancy condition, then the algorithm removes it from the policy. The algorithm ends when no more authorizations satisfy the redundancy condition w.r.t. P; this means that the algorithm has reached a fixed point, and from Theorem 3 further reductions are not possible.

In order to improve the performance of the algorithm, we can optimize the *computeRegion* and the *isRemovable* procedures, by taking into account a specific conflict resolution strategy. For instance, if we consider the conflict resolution strategy presented in Section 2.1 we can tune both procedures in the following way. Let a be an authorization and P be the policy, the *computeRegion* procedure can produce a restricted region $R'_a = A_a \cup D_a \cup I_a$ where A_a is the set of direct ancestors of a (i.e., $A_a = \{a' \in \Delta | a \preceq a' \wedge \nexists a'' \in \Delta \setminus \{a, a'\} : (a \preceq a'' \wedge a'' \preceq a')\}$), D_a is the set of direct descendants of a (i.e., $D_a = \{a' \in \Delta | a' \preceq a \wedge \nexists a'' \in \Delta \setminus \{a, a'\} : (a'' \preceq a \wedge a' \preceq a'')\}$) and I_a is the set of most specific authorizations that have an intersection (at the target level) with a (i.e., $I_a = \{a' \in \Delta | a' \npreceq a \wedge a \npreceq a' \wedge T_a \cap T_{a'} \neq \emptyset \wedge \nexists a'' \in P \setminus \{a, a'\} : (a'' \preceq a' \wedge T_a \cap T_{a''} \neq \emptyset)\}$). The region R'_a is a subset of the region R_a and usually R'_a is quite smaller than R_a. In the same way we can improve the performance of the *isRemovable* procedure by leveraging the characteristics of the conflict resolution strategy, e.g, first we may check whether the *Most Specific Wins* criterion can be applied; if this is the case, then we need only to find the most specific authorization, otherwise we know that if at least one negative authorization is in the region, then the sign of the resulting authorization will be $-$ otherwise $+$.

4.3 MPP to Weighted SAT

In order to obtain an exact solution to the *MPP* problem, we can map it to the *Weighted SAT* problem, and we can use efficient *SAT*-solvers in order to identify the minimum version of the input policy. Let $P =< \Delta, \psi >$ be the input policy, we want to produce a policy $P' =< \Delta', \psi >$ where P' is a minimum version of P. We denote with G the set *privileges*(Δ, ψ). Let k be $|G|$ and n be $|\Delta|$.

We define the following variables:

- $auth_1, \ldots, auth_n$ where each $auth_i$ represents an authorization,
- $p_{i,j}$ for each $i \in [1, n]$, $j \in [1, k]$. Each variable $p_{i,j}$ represents the fact that the privilege $p_j \in G$ is assigned to the authorization a_i,

The cost function $min\ \sum_{j \in [1,n]} auth_j$ aims at minimizing the number of authorizations in the resulting policy. We define the following clauses that aim at ensuring that the resulting policy is equivalent to the initial policy:

- For each privilege p_j we define a clause in the form $\bigvee_{i \in [1,n]} p_{i,j}$. The clause aims at enforcing the fact that the privilege has to be assigned to at least one authorization.
- For each authorization $auth_i$ we define the clauses in the form $\neg p_{i,j} \vee auth_i$ for each $j \in [1, k]$, which enforce the fact that if one of the privileges is assigned to an authorization, then the variable associated with the authorization is enabled.
- For each pair of privileges p_l and p_m such that $p_l \neq p_m$ and $sign(p_l) \neq sign(p_m)$ and $privileges(\{<\{principal(p_l), principal(p_m)\}, \{action(p_l), action(p_m)\}, \{resource(p_l), resource(p_m)\}>\}) \nsubseteq Z$ we define the clauses in the form $\overline{p_{i,l}} \vee \overline{p_{i,m}}$ for each $i \in [1, n]$ that enforce the fact that two incompatible privileges cannot be assigned to the same authorization.

The resulting policy P' is obtained by creating the authorizations $auth_i$ where $i \in [1, n]$ to which at least one privilege has been assigned to.

Algorithm 4: Heuristic Algorithm for MPP

```
Input  : Policy $P =< \Delta, \psi >$
Output: Policy $P' =< \Delta', \psi >$
begin
    $\Delta' = \emptyset$;
    $Pr = privileges(\Delta, \psi)$;
    $List\ pList = new\ List(Pr)$;
    while $pList \neq \emptyset$ do
        $p = pList[0]$;
        $pList = pList.remove(p)$;
        $added = false$;
        for $a \in P'$ do
            if $isCompatible(p, a, Pr)$ then
                $added = true$;
                $T = privileges(a)$;
                if $p \notin T$ then
                    $principals(a) =$
                    $principals(a) \cup principal(p)$;
                    $actions(a) = actions(a) \cup action(p)$;
                    $resources(a) = resources(a) \cup resource(p)$;
                    break;
        if $!added$ then
            $a =<$
            $\{principal(p)\}, \{action(p)\}, \{resource(p)\} >$;
            $\Delta' = \Delta' \cup \{a\}$;
```

4.4 Heuristic Algorithm for MPP

We defined a heuristic algorithm, shown in Algorithm 4, which iteratively tries to build the minimum policy by adding a privilege at a time. First the algorithm computes the set of privileges granted by the policy P given as input. The algorithm tries to group together compatible privileges. In order to do this it iterates over the privileges, and at each iteration it tries to add the current privilege to the already existing authorizations. If no compatible authorization exists, it creates a new authorization. The *isCompatible* procedure takes as input a privilege p, an authorization a and the set of privileges Pr granted by the original policy and checks whether a and p can be merged or not, i.e., *isCompatible(p,a,Pr)* $=$ *privileges*$(\{<$*principal*$(p)\cup$ *principals*(a), *action*$(p)\cup$ *actions*(a), *resource*$(p)\cup$ *resources*(a) $>\}) \subseteq Pr \wedge$*sign*(p) $=$ *sign*(a).

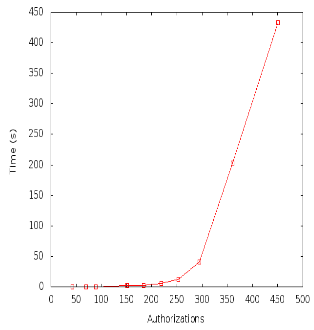


Figure 4 SAT MIPP

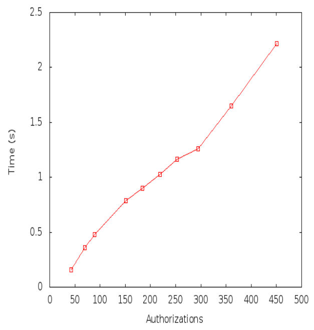


Figure 5 Heuristic MIPP

Original Size	Final Size	Delta Time %
43	22	25.58%
70	29	10.19%
88	25	3.78%
151	47	60.74%
184	58	58.86%
219	76	83.08%
253	92	90.64%
295	98	96.88%
360	112	99.19%
451	149	99.49%

Table 1 Comparison between the two MIPP methods

4.5 Experimental Results

We implemented a prototype for the evaluation of the performance of the techniques presented in this paper. The prototype consists of a Java module that invokes the implementation of the four approaches presented above. The exact solutions use the SAT4J[3] SAT solver. Since there are no freely available large datasets of real security policies, we chose to test our prototype against policies built according to an interpretation of the data in bibliographic databases. We used randomly selected subsets of *PubMed Central*[4] (PMC) which provides a rich set of attributes and relationships that represent a real and extensive social network. It has rich information about journals, with a description of editorships and the funding of papers.

Given a random sample of the *PMC* database, we built an instance of our model in the following way: (a) for each author or editor, we create a *principal*, (b) for each group of authors that have written a paper together, we create a *principal* containing the *principals* associated with the authors, (c) for each group of editors of a conference or a journal, we create a *principal* containing the *principals* associated with the groups (d) we defined three different actions *read*, *write* and *review*. We then created the following authorizations: (a) for each paper author we create the authorizations to *read* and *write* the paper and the negative authorization to *review* the paper, (b) for each editor of the issue of the journal containing the paper we add the authorizations to *read* and *review* the paper and the negative authorization to *write* the paper, (c) for each author that receives funding from the same grant that funded the paper, we add a negative authorization to *review* the paper and an authorization to *read* the paper, (d) for each group representing the institution to which the author is affiliated, we add the authorization to *read* the paper, (e) for each group representing the editorial board of the journal that published the paper we add the authorizations to *read* and *review* the paper and the negative authorization to *write* the paper. Experiments have been run on a PC with two Intel Xeon 2.0GHz/L3-4MB processors, 12GB RAM, four 1-Tbyte disks and Linux operating system. Each observation is the average of the execution of ten runs.

[3]http://www.sat4j.org - Sat4j library for Java
[4]http://www.ncbi.nlm.nih.gov/pmc/

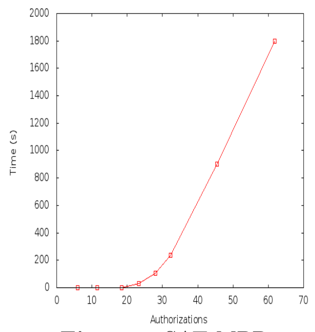


Figure 6 SAT MPP

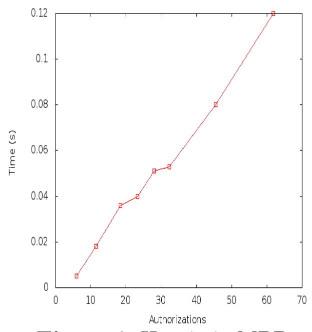


Figure 7 Heuristic MPP

Original Size	Final Size	Delta Time %
6	4	93.67%
12	4	83.94%
19	4	76.70%
23	8	99.86%
28	9	99.95%
32	10	99.98%
45	11	99.99%
62	13	99.99%

Table 2 Comparison between the two MPP methods

The results of the exact approach for the MIPP problem are shown in Figure 4, whereas the performance of the heuristic algorithm are shown in Figure 5. Table 1 shows a detailed comparison between the results of the two approaches. The heuristic approach is always able to identify the exact solution and the savings in execution time range from 3.78% to 99.49%.

The results of the exact approach for the MPP problem are shown in Figure 6, whereas the performance of the heuristic algorithm are shown in Figure 6. Table 2 shows a detailed comparison between the results of the two approaches. Also in this case the heuristic approach is always able to identify the exact solution and the savings in execution time range from 76.70% to 99.99%. However the MIPP exact solution scales better than the MPP one.

Empirical results show that both heuristic algorithms could be a good approximation of the exact ones. They allow the analysis of real policies with a good precision and with good response time. Figure 8 and Figure 9 compare the performance of the two heuristic algorithms, both in terms of execution time and reduction. The execution time of the two algorithms is very similar with policy with a size lower

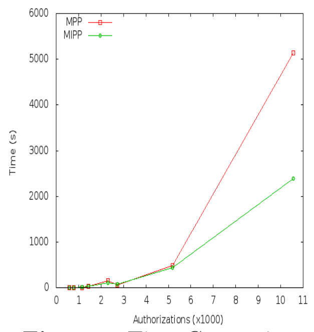


Figure 8 Time Comparison

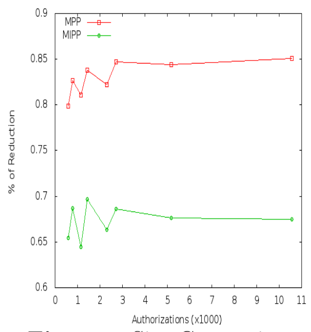


Figure 9 Size Comparison

than 5000 authorizations. After that threshold the MIPP algorithm performs better. On the other hand, the MPP algorithm allows to compute smaller policies than the MIPP. The MPP allows to obtain a policy which is usually 20% smaller than the policies resulting from the MIPP, but its execution may take more time.

5 Related Work

Several works recognize that real security policies may contain redundancy [13,16], and that this fact may lead to an increase in the total management costs of the policies. Redundancy may also reduce the efficiency of the global access control system.

The firewall community has shown a great interest in detection and removal of redundancy from firewall policies [1–3, 5, 11, 16].

Basile et al. [5] present an approach to anomaly detection which is based on the representation of policies by means of hyper-rectangles. Their approach can be used to identify redundant rules in firewall policies. Fireman [16] is a tool that can be used also to detect redundant rules in firewall policies by using Binary Decision Diagrams. Al-Shaer and Hamed proposed *Firewall Policy Advisor* [1–3], a tool that is able to detect several anomalies in firewall policies and also redundancies. In [11] Liu and Gouda model policies by means of *Firewall Decision Trees*, and present an algorithm that can be used to detect redundant rules.

In our opinion the models used in these works are quite simple, because they usually have only two actions, i.e. *accept* and *deny*, and they consider only the hierarchy of IP addresses, and thus comparison with approaches that consider more complex models may be difficult.

An interest in anomaly detection for access control policies has grown also in the computer security community [4,8–10].

Kolovski et al. [10] tackle the fact that complex XACML policies are hard to understand and evaluate manually, and, thus, automated tools are needed. They propose to map XACML to Description Logics (DL), to benefit from off-the-shelf DL reasoners. They implement in DL a reasoning service that can be used to identify redundant authorizations. Also Hu et al. [9] provide a way to detect redundant authorizations in XACML policies. They model access control policies by means of boolean expressions, and then they represent these expressions using BDDs.

Although our work does not consider explicitly a particular scenario, our model is general enough to be adapted to represent both firewall policies (there is only one *Principal* which represents any incoming packet, *Actions* are only *deny* and *accept*, and *Resources* represent IP addresses) and XACML policies (the mapping from XACML to our model is trivial in case of stateless XACML policies).

All the works presented above propose only solutions that compute an *irreducible version* of the original policy given as input, they do not try to compute the *minimum irreducible version* nor the *minimum version* of the original policy. To the best of our knowledge no algorithms for the *Minimum Irreducible Policy Problem* and *Minimum Policy Problem* exist.

Despite the fact that the *Minimum Policy Problem* may have some aspects in common with Role Mining [7], i.e., it tries to compute a minimum representation of the behaviour of the access control mechanism, there are also several differences, i.e. (a) *MPP* can be applied also to situations in which no roles exist (e.g. in firewalls), (b) *MPP* does not try to extract any meaningful information from the user-permission assignments, (c) *MPP* can be applied also as an optimization step just before the actual deployment of the policy; in this way by obtaining a smaller equivalent policy security administrators can implement a policy that achieves better performance [11,16].

6 Conclusions

We analyzed the presence of redundancies in access control policies and we have shown that the interesting problems related with redundancy are NP-hard. Although exact solutions can be found by means of *SAT solvers*, the empirical evidence we reported confirms that the approaches do not scale well. Thus, we proposed two *heuristic algorithms*. For the *Minimum Irreducible Policy Problem* we defined a heuristic approach that iteratively computes which authorizations satisfy the redundancy condition by computing their regions. For the *Minimum Policy Problem* we defined a heuristic approach that iteratively tries to identify compatible privileges. We conducted a detailed performance analysis, which shows that our heuristic solutions can compute solutions very close to the optimal one, with a performance that, as expected, significantly outperforms exact solutions in terms of execution time and remains applicable even with large policies.

7 References

[1] Al-Shaer, E., and Hamed, H. Firewall policy advisor for anomaly detection and rule editing. In *Proc. of IEEE/IFIP Integrated Management* (2003).

[2] Al-Shaer, E., and Hamed, H. Discovery of policy anomalies in distributed firewalls. In *Proc. of IEEE Infocom* (2004).

[3] Al-Shaer, E., and Hamed, H. Modeling and management of firewall policies. *IEEE Transactions on Network and Service Management 1*, 1 (2004).

[4] Arrigoni Neri, M., Guarnieri, M., Magri, E., Mutti, S., and Paraboschi, S. Conflict detection in security policies using semantic web technology. In *Proc. of the ESTEL 2012 - Security Track*.

[5] Basile, C., Cappadonia, A., and Lioy, A. Network-level access control policy analysis and transformation. *Networking, IEEE/ACM Transactions on*, 99 (2011).

[6] Bonatti, P., De Capitani di Vimercati, S., and Samarati, P. An algebra for composing access control policies. *ACM TISSEC 5*, 1 (2002).

[7] Frank, M., Buhmann, J. M., and Basin, D. On the definition of role mining. In *Proc. of SACMAT 2010*, ACM.

[8] Guarnieri, M., Magri, E., and Mutti, S. Automated management and analysis of security policies using eclipse. In *Proc. of the Eclipse-IT 2012*.

[9] Hu, H., Ahn, G., and Kulkarni, K. Anomaly discovery and resolution in web access control policies. In *Proc. of SACMAT '11*, ACM.

[10] Kolovski, V., Hendler, J., and Parsia, B. Analyzing web access control policies. In *Proc. of WWW 2007*, ACM.

[11] Liu, A., and Gouda, M. Complete redundancy detection in firewalls. In *Data and Applications Security XIX*, vol. 3654. 2005.

[12] Lupu, E., and Sloman, M. Conflicts in policy-based distributed systems management. *IEEE TSE 25*, 6 (1999).

[13] Molloy, I., Chen, H., Li, T., Wang, Q., Li, N., Bertino, E., Calo, S., and Lobo, J. Mining roles with multiple objectives. *ACM TISSEC 2010*.

[14] Sandhu, R. Role-based access control. *Advances in computers 46* (1998).

[15] Shen, H., and Dewan, P. Access control for collaborative environments. In *Proc. of the CSCW'92*, ACM.

[16] Yuan, L., Chen, H., Mai, J., Chuah, C., Su, Z., and Mohapatra, P. FIREMAN: a toolkit for firewall modeling and analysis. In *Proc. of IEEE S&P* (2006).

APPENDIX

Lemma 1. Let $P =< \Delta, \psi >$ be a policy and let *auth* an authorization in Δ. If *auth* satisfies the *redundancy condition* w.r.t. P then *privileges*(Δ, ψ) $=$*privileges*$(\Delta \setminus \{auth\}, \psi)$.

Proof of Lemma 1 We prove the lemma by contradiction. We assume that *auth* satisfies the *redundancy condition* w.r.t. P, and that *privileges*(Δ, ψ) $\neq$*privileges*$(\Delta \setminus \{auth\}, \psi)$. This means that there is at least a target $< p, a, r >$ that behaves in a different way under P and under $P \setminus \{auth\}$. This fact implies that $\exists s \in \{+, -\}$: $< s, p, a, r > \in$*privileges*$(auth)$ such that one of the two following cases holds:

1. $< s, p, a, r > \in$*privileges*(Δ, ψ) and does not exist any sign $s' \in \{+, -\}$ such that $< s', p, a, r > \in$ *privileges*$(\Delta \setminus \{auth\}, \psi)$. This means that $\psi(p, a, r, \Delta \setminus \{auth\}) =\perp$, but this generates a contradiction due to the *completeness* property of ψ and the definition of the *redundancy condition.*
2. $< s, p, a, r > \in$*privileges*$(\Delta, \psi) \wedge$ $< s', p, a, r > \in$*privileges*$(\Delta \setminus \{auth\}, \psi)$ where s' is the opposite sign of s (i.e., if $s = +$ then $s' = -$ and viceversa), but this fact contradicts the definition of the *redundancy condition.* $\Box$

Proof of Theorem 1 It follows trivially from Lemma 1. $\Box$

Proof of Theorem 2 We prove the theorem by contradiction. We assume that there is an *auth* that satisfies the redundancy condition w.r.t. P_j but does not satisfies the redundancy condition w.r.t. a $P_i \cup \{auth\}$ where $j < i \leq n$. Due to the fact that $R_{auth}^{P_i \cup \{auth\}} \subseteq R_{auth}^{P_j}$, it follows that there is at least a privilege $< s, p, a, r > \in$*privileges*$(auth)$ for which one of the following holds:

1. $\psi(p, a, r, R_{auth}^{P_i \cup \{auth\}}) =\perp$, but this is impossible because $R_{auth}^{P_i \cup \{auth\}}$ contains *auth* and ψ is complete.
2. *sign*$(\psi(p, a, r, R_{auth}^{P_i \cup \{auth\}} \setminus \{auth\})) \neq$ *sign*$(\psi(p, a, r, R_{auth}^{P_i \cup \{auth\}}))$ (we refer in the following to this property as $\circ$). From the fact that all the authorizations that are in P_j but not in $P_i \cup \{auth\}$ satisfies the redundancy condition, it follows that also if we add them to $P_i \cup \{auth\}$ they preserve the $\circ$ property. This means that also $P_i \cup \{auth\} \cup (P_j \setminus (P_i \cup \{auth\})) = P_j$ satisfies the $\circ$ property, but this is a contradiction.

In both cases we have a contradiction, so we have proved the theorem. $\Box$

Lemma 2. Let P be a security policy. There is a $P' \subset P$ equivalent to P iff there is at least an authorization $auth \in P$ that satisfies the redundancy condition w.r.t. P.

Proof of Lemma 2 $\Leftarrow$) Suppose there is an authorization $auth \in P$ that satisfies the redundancy condition w.r.t. P. This means that there is a sequence of reductions in which P and $P' = P \setminus \{auth\}$ appear, and thus from Theorem 1 we know that P and P' are equivalent (and $P' \subset P$ holds). $\Rightarrow$) We prove this part by contradiction. Let $P' \subset P$ be equivalent to P and let $S \subseteq P$ be the set of authorizations that satisfies the redundancy condition w.r.t. P. We assume that $S = \emptyset$. From the fact that $P' \subset P$ and P' is equivalent to P, follows that all the authorizations in $K = P \setminus P'$ do not influence the behaviour of P. Without loss of generality, we consider only the case in which $|K| = 1$ (i.e., $K = \{auth'\}$). The fact that $auth'$ does not influence the behaviour of P means that for each privilege $< s, p, a, r > \in$*privileges*$(auth')$ the following statements hold:

- *sign*$(\psi(p, a, r, P \setminus \{auth'\}))$ = *sign*$(\psi(p, a, r, P))$ and from the fact that ψ satisfies the monotonicity property this is equivalent to *sign*$(\psi(p, a, r, R_{auth'}^{P} \setminus \{auth'\}))$ = *sign*$(\psi(p, a, r, R_{auth'}^{P}))$,
- $\psi(p, a, r, P \setminus \{auth\}) \neq \perp$, but since ψ satisfies the completeness property this implies the fact that $\psi(p, a, r, R_{auth'}^{P} \setminus \{auth\}) \neq \perp$.

This means that $auth'$ is redundant w.r.t. P, and thus $S \neq \emptyset$ which lead to a contradiction. $\Box$

Proof of Theorem 3 It follows trivially from Lemma 2. $\Box$

Algorithm 5: *irreduciblePolicy* procedure

```
Input  : Policy $P =< \Delta, \psi >$
Output: Policy $P' =< \Delta', \psi > where P' \subseteq P$
begin
    $\Delta' = \Delta$;
    bool continue =true;
    while continue do
        $\Gamma = \emptyset$;
        continue =false;
        for $auth \in \Delta'$ do
            if $satisfies(auth, \Delta')$ then
                $\Gamma = \Gamma \cup \{auth\}$;
                continue =true;
                Break;
        $\Delta' = \Delta' \setminus \Gamma$;
```

Proof of Theorem 4 Algorithm 5 computes an irreducible version of the policy P given as input, and it can be executed in polynomial time (the procedure *satisfies*$(auth, \Delta')$ checks whether the authorization *auth* satisfy the redundancy condition w.r.t. Δ' and it can be executed in time polynomial in the size of the set Δ and in the size of the Target Hierarchy). Given a policy P we can check if P is irreducible by checking whether P = *irreduciblePolicy(P)*. $\Box$

Proof of Theorem 5 A nondeterministic algorithm need only to guess a sequence of reduction S starting from P'. If P is not an irreducible version of P', then the solution of $\overline{MIPP}$ is trivial. Otherwise if $|end(S)| < |P|$, then we have found an irreducible version of the policy P' with less authorizations than P, and thus P is not a Minimum Irreducible Policy and we have found a solution to the $\overline{MIPP}$. We have already shown that the $\overline{MIPP}$ is in **NP**. Now we have to show that a **NP-complete** problem can be reduced in polynomial time to $\overline{MIPP}$. We consider the *Set Covering Problem* (SCP).The *SCP* is defined in the following way: given a set X, a collection of subsets of X called C, such that $X = \bigcup_{C_i \in C} C_i$, and an integer value $k < |C|$, does exist a $C' \subset C$ such that $X = \bigcup_{C'_i \in C'} C'_i$ and $|C'| \leq k$?

We can map *SCP* to $\overline{MIPP}$ in the following way:

- we define an action a and a resource r,
- for each element $x_i \in X$ we define a principal p_i and a privilege $< +, p_i, a, r >$,
- for each element $C_i \in C$ we define an authorization $< +, P_i, a, r >$ where P_i is the set containing all the principals associated with the elements $x_j \in C_i$,
- we define the original policy P' composed by all the authorizations associated with each $C_i \in C$. The policy P' has $|C|$ authorizations,
- we define the policy P composed by $k+1$ authorizations in the following way: it contains the k authorizations associated with $C_1, \ldots, C_k$, it contains an authorization $< +, P_{all}, a, r >$ where P_{all} contains all the principals associated with the elements in $C_{k+1}, \ldots, C_{|C|}$.

The algorithm presented above is obviously polynomial, and it produces a pair of policies. The original policy P' is composed by $|C|$ authorizations, each one with the form $auth_i =< +, \{p_1, \ldots, p_j\}, a, r >$ where $p_1, \ldots, p_j$ are the principals associated with the elements of X which belong to C_i. The policy P which has to be checked for minimum irreducibility is composed by $k+1$ authorizations.

We have only to show that $SCP \Longleftrightarrow \overline{MIPP}$. On the one hand we assume that there exists a $C' \subset C$ such that $X = \bigcup_{C'_i \in C'} C'_i$ and $|C'| \leq k < |C|$. This means that we can create a policy P'' composed by $|C'|$ authorizations (we generate an authorization $auth_i$ that contains all the privileges associated with the elements of C'_i for each element $C'_i \in C$) that is equivalent to P', because the privileges in P'' and P' are the same given the fact that C' is a cover for X. In addition, P'' is smaller than P ($|P''| \leq k \leq |P|$) and $P'' \subset P'$, and thus we have found an equivalent policy smaller than P. Also the $\overline{MIPP}$ evaluates to *yes* because P is not a minimum irreducible policy. Conversely, we assume that $\overline{MIPP}$ evaluates to *yes* and thus the policy P is not a minimum irreducible version of P'. This means that there exists a policy P'', equivalent to P', such that $|P''| < |P| = k+1$ and $P'' \subseteq P'$ and this means that there exists a cover $C' \subset C$ with the same cardinality of P'' and thus $|C'| \leq k$. □

Given a set of authorizations A and an authorization *auth*, we can compute whether *auth* implies all the authorizations in the set A by means of the function *implies(auth, A)* shown in Algorithm 6 (the functions *principals+*, *actions+* and *resources+* are the transitive closures of *principals*, *actions* and *resources* over the respective hierarchies).

Proof of Theorem 6 A nondeterministic algorithm need only to guess a set of authorizations A, and an authorization *auth* such that $|A| \geq 2$, $A \subseteq P$ and $auth \notin A$, and then

Algorithm 6: *implies* procedure

```
Input  : $auth$, $A$
Output: $True$ or $False$
begin
    for $auth' \in A$ do
        if $sign(auth) = sign(auth')$ then
            if $principals+(auth') \not\subseteq principals+(auth) \vee actions+(auth') \not\subseteq actions+(auth) \vee resources+(auth') \not\subseteq resources+(auth)$ then
                return $False$;
        else
            return $False$;
    return $True$;
```

we can check in polynomial time, using the Algorithm 6, whether the policy is not minimum. If the *implies* procedure returns *True*, this means that an equivalent policy P' with a size of $|P| - |A| + 1$ exists (this policy is the one that we can obtain by removing from P all the authorizations in A and adding *auth*) and thus we have found a solution to the $\overline{MPP}$ (i.e., given a policy P checking whether P is not minimum). We have already shown that the $\overline{MPP}$ is in **NP**. Now we have to show that an **NP-complete** problem can be reduced in polynomial time to $\overline{MPP}$. We consider the *Set Covering Problem* (SCP).The *SCP* is defined in the following way: given a set X, a collection of subsets of X called C, such that $X = \bigcup_{C_i \in C} C_i$, and an integer value $k < |C|$, does exist a $C' \subset C$ such that $X = \bigcup_{C'_i \in C'} C'_i$ and $|C'| \leq k$?

We can map *SCP* to $\overline{MPP}$ in the following way:

- we define an action a and a resource r,
- for each element $x_i \in X$ we define a principal p_i and a privilege $< +, p_i, a, r >$,
- we define a policy composed by $k+1$ authorizations $auth_0, \ldots, auth_k$,
- for each element $C_i \in C$ we add the privileges associated with all the elements in C_i to the authorization $auth_{i \ mod(k+1)}$.

The algorithm presented above is obviously polynomial, and it produces a policy composed by $k+1$ authorizations, each one with the form $auth_i =< +, \{p_1, \ldots, p_{l_i}\}, a, r >$ where l_i is the number of principals in the authorization $auth_i$.

We have only to show that $SCP \Longleftrightarrow \overline{MPP}$. On the one hand we assume that there exists a $C' \subset C$ such that $X = \bigcup_{C'_i \in C'} C'_i$ and $|C'| \leq k$. This means that we can create a policy P' composed by $|C'|$ authorizations (we generate an authorization $auth_i$ for each element $C'_i \in C$ that contains all the privileges associated with the elements of C'_i) that is equivalent to P, because the privileges in P and P' are the same given the fact that C' is a cover for X. In addition, the size of P is at most k that is smaller than P, and thus also the $\overline{MPP}$ evaluates to *yes*. Conversely, we assume that $\overline{MPP}$ evaluates to *yes* and thus the policy P is not minimum. This means that there exists a policy P', equivalent to P, such that $|P'| < |P| = k+1$ and thus this means that there exists a cover $C' \subset C$ with the same cardinality of P' and thus $|C'| \leq k$. □