Policy Groups

Winfried E. Kühlhauser
GMD National Research Center For Information Technology
D-53754 Sankt Augustin, Germany

This paper contributes to the current discussion on multipolicy systems: Systems that support a multitude of independent security domains in which an individual security policy is enforced on the applications. In multipolicy systems, the interoperability between different security domains constitutes a major problem. While security policies are capable of controlling the applications within their domains, interactions between different security domains create security loop-holes and cause conflicts between the involved security policies.

This paper introduces policy groups as an approach to secure domain interactions. A policy group combines a set of security policies with a set of policies that control interdomain actions. It composes a multipolicy system's security policies into a single structure and provides a single point of reference for the discussion of a system's security properties.

In order to provide a precise foundation for this discussion, the paper introduces a formal model of policy groups based on Harrison, Ruzzo and Ullman's access control calculus. The paper concludes with statements about the decidability of the safety problem for policy groups.

Keywords: security policy, multipolicy system, information domain, policy domain, policy group.

1 Introduction

In large organizations, many branches and departments have their own unique information security requirements. Mapping the structure of such organizations to a computer system results in a distributed system with a multitude of information security domains, each security domain having its individual and unique security policy.

In 1993, the trend towards multipolicy systems was taken up by the U.S. Department of Defense. As part of DoD's Technical Architecture Framework for Information Management (the current version is [2]) a corresponding set of information security requirements for multipolicy systems was identified. These requirements state that DoD information systems must [4]

- support information processing under multiple security policies of any complexity or type, including those for sensitive unclassified information and multiple categories of classified information
- support information processing among users with different security attributes employing resources with varying degrees of security protection, including users of non-secure resources.

These requirements imply new challenges in many areas of system design. As an example, in order to protect security policies from influencing each other, system designers must solve the problem of policy separation - a problem that is not covered by the traditional reference monitor approach. Another challenge is the enforcement and separation of policy domains:
coexisting but independent protected environments where a single security policy is enforced on any user, application, or resource.

While these challenges primarily concern operating system designers (see Edward Feustel’s and Terry Mayfield’s paper on Unmet Information Security Challenges for Operating System Designers [4]), our concern in this paper are the challenges for security policy designers. Although multiple security policy domains provide a clear concept to represent different security strategies within the same system, coexisting policy domains are a threat to a system’s interoperability; they tend to establish autonomous islands that are protected by their security policy, and it is unclear how trips between the islands can be made and managed with respect to security.

Any such trip between security policy domains can be highly security critical. Consider two policies, one being a simple discretionary access control policy (such as in Unix), the other being a mandatory multilevel policy. Any interaction between the domains would require the enforcement of well thought-out rules describing precisely the conditions under which a subject from the DAC domain may access an object within the MAC domain – or vice versa. Other related examples of the general interdomain actions problem are the Propagation Of Local Risk Problem and the Cascading Problem identified in [12].

This paper approaches secure interdomain actions by a concept we call policy groups. A policy group is a set of regular security policies together with a set of policies that control interdomain actions, and a classification function that for any given interdomain action selects the "right" policy from the sets. The sensitive role of policy groups implies a concern for their formal analysis. Section 3 provides a precise definition of a policy group in terms of the well-known Harrison/Ruzzo/Ullman (HRU) access control calculus [7, 8] and the interdomain actions classification of [9]. The (HRU-) safety properties of policy groups are discussed in section 4.

2 A Classification of Interdomain Actions

This section is a brief recap of [9]. In order to precisely define different classes of actions in multipolicy systems, [9] applies Lampson’s traditional approach to access control modelling in which any interaction between the entities of a system is modelled as a subject \(s\) accessing an object \(o\) via some access operation \(a\). \(P(s,o,a)\) denotes the application of policy \(P\) to a specific access \((s,o,a)\); \(P\) is an access control policy, so \(P(s,o,a)\) is of type \{granted, rejected\}. We use \(Dom_P\) to denote the domain belonging to \(P\) which consists of all entities (subjects and objects) that are submitted to \(P\). For any access \((s,o,a)\), a policy \(P\) will contain an access rule if and only if \(s,o \in Dom_P\). \(\Pi_e = \{P|e \in Dom_P\}\) denotes the set of security policies that have entity \(e\) within their domain. Last but not least we use \(|M|\) as the usual denotation for the cardinality of some set \(M\).

Let \(I\) be a finite set of indices, \(\{P_i\}_{i \in I}\) a set of security policies, \(s,o \in \bigcup_{i \in I} Dom_{P_i}\), and \(a\) some access operation. In any multidomain scenario with a set of security policies \(\{P_i\}_{i \in I}\), any access \((s,o,a)\) belongs to one of the following three classes.
Class 1: $|\Pi_s| = |\Pi_o| = 1 \land \Pi_s = \Pi_o$

Actions of class one are characterized by the situation that subject and object are members of the same domain and are not a member of any other domain. Actions within this class do not cross domain borders, and consequently, a single policy is both capable and authorized to make the access decision.

Class 2: $|\Pi_s \cap \Pi_o| = 0$

This access class is characterized by the situation that no security policy exists that has both subject and object in its domain. Especially, there is no security policy that is capable of providing a rule for this particular access. Nevertheless, because $s, o \in \bigcup_{i \in I} Dom_R$, there are security policies that encompass one of the entities within their domain. An example for a class 2 access is shown in figure 2 in the following section.

Class 3: $|\Pi_s \cap \Pi_o| \geq 1 \land \exists e \in \{s, o\} : |\Pi_e| > 1$

This access class is characterized by the situation that on the one hand there is (at least) one policy that might provide a rule for the access; nevertheless, on the other hand (at least) one of the involved entities is a member of more than one domain. Examples for class 3 accesses are shown in figures 3 and 4.

This classification is both unique and complete: any access $(s, o, a)$ in a multipolicy system with $s, o \in \bigcup_{i \in I} Dom_R$ belongs to exactly one class (proof cf. [9]). [9] now argues that this classification provides precise necessary and sufficient conditions to identify those accesses in multipolicy systems that cannot be controlled by the system’s regular security policies. The paper concludes with the identification of two new roles of security policies in a multipolicy system: the mending of loop-holes and the mediation of conflicts.

3 Policy Groups

Within this paper, we take up these results and carry them further towards an approach to deal with loop-holes and conflicts in multipolicy systems.

Considering the above interdomain action classification, for any class-two action we are stuck with the fact that none of the existing security policies yields a rule for the action. Although there might be more than one domain that contains either the subject or the object there is no domain that contains both entities. Consequently, in order to restore well-defined information security properties in this situation, additional rules that implement interdomain security requirements have to be furnished. In other words, class-two accesses require an additional completeness policy that closes this hole.

For any class-three interdomain action we are stuck with the fact that at least one security policy yields a rule for the action, and at least one of the involved entities is a member of at least one other policy domain. This accounts for two different types of conflict. If $|\Pi_s \cap \Pi_o| = 1$, there exists exactly one policy yielding a rule for the particular access; however this rule may not be applied alone because one of the entities belongs to a second domain. We refer to this type of conflict as a domain conflict. If $|\Pi_s \cap \Pi_o| > 1$, then there
exists more than one policy that yields a rule for the particular access. Additionally there might be more policies that encompass one of the entities in their domain. We refer to this type of conflict as a rule conflict. Both types of conflicts cannot be mediated by the existing security policies alone. Additional security requirements that regulate conflict resolution have to be furnished, requiring an additional conflict mediation policy.

In the following definition, we combine a multipolicy system's regular security policies, completeness policy and conflict mediation into a single policy group.

**Definition (Policy Group)**

Let $I$ be a finite index set and $\{P_i\}_{i \in I}$ the set of regular security policies of a given multipolicy system. A policy group $G$ is a tuple $G = (\{P_i\}_{i \in I}, T, F, c)$, consisting of

- a finite set of regular security policies $\{P_i\}_{i \in I}$, implementing the security requirements for class-one accesses
- a completeness policy $T$, implementing the security requirements for class-two accesses
- a conflict mediation policy $F$, implementing the security requirements for class-three accesses
- a classification function $c$ that for each access $(s, o), s, o \in \bigcup_{i \in I} \text{Dom}_{P_i}$, yields the class of $(s, o)$.\(^1\)

$T$ and $F$ are implemented in exactly the same paradigms and are enforced with exactly the same mechanisms as any regular security policy of a multipolicy system. However, in contrast to any regular security policy, the domains of $T$ and $F$ encompass the domains of every single regular security policy: $\text{Dom}_T = \text{Dom}_F = \bigcup_{i \in I} \text{Dom}_{P_i}$.

The classification function $c$ is of type $\bigcup_{i \in I} \text{Dom}_{P_i} \times \bigcup_{i \in I} \text{Dom}_{P_i} \to \{P_i\}_{i \in I} \cup T \cup F$ and is for any $s, o \in \bigcup_{i \in I} \text{Dom}_{P_i}$ defined by

$$
c(s, o) = \begin{cases} 
P_k & |\Pi_s| = |\Pi_o| = 1 \quad \Pi_s = \Pi_o = \{P_k\} \\
T & |\Pi_s \cap \Pi_o| = 0 \\
F & |\Pi_s \cap \Pi_o| \geq 1 \quad \exists e \in \{s, o\} : |\Pi_e| > 1 
\end{cases}
$$

As argued in [9], the classification function becomes part of the system’s reference monitor that implements the total access mediation property [3] by overwriting the regular security policy call that is issued on every entity interaction. While any class-one interaction is directed to its regular security policy, class-two and class-three interactions are diverted to $T$ respectively $F$.

**An Example**

Let us now look at an example that illustrates this definition. Consider a large organization with many branches, each branch organized in a hierarchy of departments and projects. Any global security policy for such an organization will be unable to take into account the unique

\(^1\)As the classification depends only on $s$ and $o$, we will omit $a$ here
security requirements of every individual department or project and will enforce only the most
general security requirements. As a consequence, departments and projects will usually define
their own individual security policies reflecting their specific security requirements. These
policies for example are refinements or "stronger" instances of the embracing policy in the
sense that they apply a more restrictive access control scheme. Within the organization's
computer system, the policy domain structure will then reflect the organizational hierarchy,
where the outer shell reflects the domain of the global security policy and the innermost levels
protect the most secret research & development projects, using for example mandatory access
control policies and cryptosystems that are totally incompatible with the rest of the system.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{diagram.png}
\caption{A hierarchical nesting of security policy domains}
\end{figure}

In order to improve cooperation between its departments our organization encourages its
employees to transfer their place of work temporarily to other departments. As an example,
experts from the research department may move for a few weeks to the consulting department
in order to support a fast and efficient training of the consultants. Also, a sales manager
from the marketing department may move to the development department to become familiar
with a new product, or experts from product development may move to the organization's
country-specific sites to help their colleagues with the customization of a new product.

Now let us turn to the security aspects of such temporary assignments. Let us assume that an
employee, let us call her Ann, has a temporary assignment to the consulting department that
has a security policy \( C \) in place. When Ann logs into a system of her temporary department
she then becomes a principal within \( \text{Dom}_C \). Any access of Ann to one of her own files within
her home domain \( \text{Dom}_Q \) now is an interdomain access.

For any such access the authorization scheme of our multipolicy scenario is incomplete: neither

\footnote{Firstly, because of the individual security requirements of the departments a global, organization-wide
authentication scheme will in general be inadvisable. Secondly, an authentication by the authentication server
of her home department is equally inadvisable because authentication secrets of Ann (such as her password)
would be disclosed within \( \text{Dom}_C \). Thirdly, even if her home authentication uses a challenge/response protocol
based on smartcards, \( \text{Dom}_C \) may lack the necessary hardware infrastructure.}
Figure 2: A non-hierarchical nesting of security policy domains

policy C’s nor policy Q’s authorization schemes contain a rule for Ann, principal of DomC, accessing Ann’s file, principal of DomQ, because Q does not know about Ann and C does not know about Ann’s files in DomQ.

In order to close this gap we create a policy group. The involved regular policies are C and Q (let us for a few minutes ignore the global policy E and the branch-specific policy R&D), so our policy group will be ({C, Q}, T, F, c). \( \Pi_{Ann} \) is \{C\}, and \( \Pi_{Ann’s\ file} \) is \{Q\}. The classification function \( c \) classifies Ann’s file access \( (Ann, Ann’s\ file) \) as a class-two access (because \( |\Pi_{Ann} \cap \Pi_{Ann’s\ file}| = |\{C\} \cap \{Q\}| = 0 \)). Consequently, the policy group’s completeness policy T is selected for Ann’s file access.

The policy T now implements the organization’s strategy for interdomain accesses. One such strategic rule for example is that "any person with a temporary assignment will keep all access rights from her/his home domain and will additionally obtain the rights of an employee of her/his temporary assignment”. Implementing this rule within T will use a list of all current temporary assignments, in our case mapping the principal Ann within DomC to the principal Ann within DomQ. This mapping may also include the adaption to different security standards within the domains; for example a renewed authentication can be demanded from Ann that in return provides the cryptographic keys and tickets that Ann will need to communicate with her home file server.

Let us now look at a situation where an interdomain access causes a conflict between the involved regular policies. In the above example, we ignored that all entities in DomC as well as in DomQ additionally belong to the domain of a global policy E, and all entities in DomQ further belong to the domain of policy R&D (see figure 2). Considering this, an appropriate policy group is \((\{E, C, R&D, Q\}, T, F, c)\): \( \Pi_{Ann} \) is \{E, R&D, Q\} when she is at home and \{E, C\} when she works in Consulting. As \( \Pi_{Ann’s\ file} \) is \{E, R&D, Q\}, \( |\Pi_{Ann} \cap \Pi_{Ann’s\ file}| \) is in both cases \( \geq 1 \), and each \( \Pi \) contains more than one policy. \( c \) thus classifies any access of Ann to her file as a class-three access.

Again, the role of policy F reflects the organization’s strategic decisions how to handle conflicts in interdomain accesses. One option for the strategists is to apply exactly the same strategy as for incompleteness; referring to the above example, this would be achieved by defining \( F = T \). A more fine-grained approach is to distinguish within \( F \) between rule conflicts \( (|\Pi_r \cap \Pi_o| > 1) \) and domain conflicts \( (|\Pi_r \cap \Pi_o| = 1) \). If Ann accesses her file while at home (figure 3), a rule conflict would occur between policies E, R&D, and Q. Considering that in our organization the innermost policies are the strongest, an adequate conflict strategy now would be that in case of a rule conflict the strongest policy will make the access decision. F would implement this strategy by passing Ann’s request to policy Q.
If on the other hand Ann accesses her files from domain $C$ (figure 4), $F$ would determine a domain conflict. In this case, $F$ could for example pass the access decision to $E$ (implementing the strategy that whenever no inner policy provides an access rule, the global policy $E$ helps out). As another example, to implement the same scheme as described above for $T$ $F$ can use a list of temporary assignments to map principal Ann in $C$ to principal Ann in $Q$.

4 Safety Analysis of Policy Groups

The security-critical nature of interdomain actions implies a major concern for a formal analysis of the governing security policies. One of the most basic properties of any access control policy is the way it guards the propagation of rights. Given some policy $P$, we want to know if

"given some initial state of $P$, will a certain subject ever be capable of gaining a certain access right to a certain object?"

This problem has become known as the safety-problem of access control policies. The safety problem of access control policies has a significant importance in practice: knowing the safety properties of an access control policy will put us in the position to make general statements
about the effects of a policy. As an example, solving the safety problem for a given hospital security policy will allow us to determine whether a staff member of the hospital’s pharmacy will ever be capable of acquiring the right to change a patient’s medical files.

This section aims at analyzing the safety problem for policy groups. Given a policy group $G$, we want to answer questions such as

"given some initial state of $G$, can it ever happen that a staff member of department $C$ can get access to Ann's files in department $Q$ during Ann’s temporary assignment?"

In 1975, Michael Harrison, Walter Ruzzo and Jeffrey Ullman published a calculus that aimed at formalizing the safety problem and at analyzing the complexity of solving it [7, 8]. Unfortunately, a major result was that the safety problem is undecidable in general. Fortunately, work that followed identified several subclasses of the general HRU-model in which the safety problem was decidable [6, 11, 1, 13]. Within this section, we take up these results and lift them to policy groups. We start with a brief recap of the precise meaning of the term ”HRU-safety” and summarize major results of the follow-up work. We then identify conditions under which safety properties of the regular security policies $P_i$ can be lifted to a policy group over the $P_i$’s.

### 4.1 HRU-Safety

The HRU security model introduced in [7, 8] is a simple formalism to define access control policies and to analyze their safety properties. The formalism is based on Lampson’s access control model [10] and uses a state machine to model a policy’s dynamic behavior. Each state is a triple $(S, O, M)$, where $S$ is a set of subjects, $O$ is a set of objects, and $M$ is an access matrix $M : S \times O \rightarrow \wp(R)$ that in each cell $M(s,o)$ contains the access rights of $s$ with respect to $o$, taken from a finite right set $R$ ($\wp$ denotes the powerset).

The dynamic behavior is modelled by a transition function $\delta$ that is defined in terms of six HRU primitives, such as enter right $r$ into matrix cell $(s,o)$ (see Appendix for a complete list). In order to model more complex application operations, these primitives can be combined in the general form

\[
\text{IF } r_1 \in M(x_{s_1}, x_{o_1}) \text{ AND } \ldots \ldots \ldots \ldots r_m \in M(x_{s_m}, x_{o_m}) \text{ THEN } p_1; \ldots; p_n \text{ FI}
\]

where $r_i$ are rights, $x_i$ are the involved subjects and objects, and $p_i$ are the mentioned HRU primitives (see appendix for a precise definition).

A single state $q^0 = (S^0, O^0, M^0)$ is now called safe for a right $r \in R$ if and only if there is no sequence of states that, beginning with $q^0$ writes $r$ into some matrix cell that did not already contain it (see appendix for a precise definition).
A major result of [7, 8] was that no general algorithm exists that determines whether a state is safe or not; the safety problem is undecidable in general. This is basically due to the fact that the general HRU model is expressive enough to model any given Turing machine. However, limiting the expressiveness of the general model in an intelligent, application-oriented way has identified useful subclasses of the general HRU model that balance expressive power with the complexity of safety analysis.

**Monooperational HRU Models**

One such subclass are the monooperational HRU models. Monooperational models allow only a single primitive within the operations part of a composition, so all compositions have the general form

\[
\text{IF } r_1 \in M(x_{s_1}, x_{o_1}) \text{ AND } \\
... \quad r_m \in M(x_{s_m}, x_{o_m}) \\
\text{THEN } \\
p_1 \\
\text{FI.}
\]

While the safety problem for monooperational models is generally decidable [8], their expressive power is rather poor. For example, they fail to model even the simple scheme of a Unix file creation where creating a file and granting the initial access rights is combined in a single atomic operation.

**Static Models**

A second subclass with a decidable (though np-complete) safety problem are static models in which the subject and object sets do not grow (i.e. there are no create-primitives). This class is more interesting than monooperational models because many hard real time systems have a static behavior and thus can be controlled by static models.

**Monotonic Monoconditional Models**

In monotonic monoconditional models, the conditional part of a primitives composition has only a single condition. Furthermore, no subject or object is ever destroyed nor can an existing right be withdrawn. The scope of monotonic monoconditional models are applications with a Bell/LaPadula-style security policy and a static classification function. In these policies, the subject and object sets as well as their classifications never change. The Bell/LaPadula basic security theorem then tells us that the access matrix is static as well.

Typical applications of this kind are for example document archives or digital libraries (being monotonic) running under Unix (having a monoconditional protection scheme). However, non-monotonic events (such as the retirement of the archivist) are outside the scope of this subclass.
4.2 Policy Group Safety

Within this section we discuss the safety problem for policy groups. Given a set of regular security policies \( \{P_i\}_{i \in I} \) in a multipolicy system, we will argue that safety properties of a policy group over the \( P_i \) can be derived from the safety properties of the individual \( P_i \). We start with defining our notion of a HRU model for a policy group. Next, we give an interpretation of the safety problem within this context. Finally, statements are made and proved that lift safety decidability properties from the \( P_i \) to policy groups over the \( P_i \).

In the following we assume that all our security policies – the \( P_i, T \) and \( F \) – are HRU-modelled. Thus, a state machine as described in the HRU model definition (see Appendix) exists for each policy group over the \( P_i \).

We now combine the state machines of all the policies in a policy group into a single new state machine in the following way.

**Definition (HRU-Model of a policy group)**

Let \( I \) be a finite index set. Let further each \( P_i \) be a HRU-modelled security policy with a model \((Q_i, E_i, \delta_i, q_i^0)\), \( Q_i = (S_i, O_i, M_i) \), as defined in the appendix. Finally, let also \( T \) and \( F \) be HRU-modelled security policies with corresponding models. The HRU-model of a policy group \( G = \{P_i\}_{i \in I} \) is a state machine \((Q_G, E_G, \delta_G, q_G^0)\) where

\[
Q_G = (S_G, O_G, M_G) \text{ is the set of states where}
\]

\[
S_G = \bigcup_{i \in I} S_i \text{ is the set of subjects},
\]

\[
O_G = \bigcup_{i \in I} O_i \text{ is the set of objects},
\]

\[
M_G : S_G \times O_G \to \wp(R_G) \text{ is an access matrix over the set of rights } R_G = \bigcup_{i \in I} R_i
\]

where \( M_G \) is defined by

\[
M_G(s, o) = \begin{cases} 
M_j(s, o) & \text{if } \alpha(s,o) = P_j \\
M_T(s, o) & \text{if } \alpha(s,o) = T \\
M_F(s, o) & \text{if } \alpha(s,o) = F,
\end{cases}
\]

\[
E_G = \bigcup_{i \in I} A_i \times (S_G \cup O_G)^k \text{ is the set of inputs,}
\]

\[
\delta_G : Q_G \times E_G \to Q_G \text{ is the transition function defined by}
\]

\[
\delta_G(q_G, (a, x)) = \begin{cases} 
\delta_j(q_j, (a, x)) & \text{if } c^k(x) = P_j \\
\delta_T(q_T, (a, x)) & \text{if } c^k(x) = T \\
\delta_F(q_F, (a, x)) & \text{if } c^k(x) = F,
\end{cases}
\]

\[
q_G^0 = (S_G^0, O_G^0, M_G^0) \text{ is the initial state defined by}
\]

\[
S_G^0 = \bigcup_{i \in I} S_i^0
\]

\[
O_G^0 = \bigcup_{i \in I} O_i^0
\]

\[
M_G^0(s, o) = \begin{cases} 
M_j^0(s, o) & \text{if } \alpha(s,o) = P_j \\
M_T^0(s, o) & \text{if } \alpha(s,o) = T \\
M_F^0(s, o) & \text{if } \alpha(s,o) = F.
\end{cases}
\]

This definition may look complicated at first sight; however, it actually describes a straightforward composition of the involved state machines where places of incompleteness and conflict
in the composed machine are identified by our well-known classification function \( c \), and are handled by the state machines of \( T \) and \( F \).

Because in the HRU calculus more than two subjects and objects can be involved in one operation, we need to define our classification function \( c \) for \( k \) dimensions. So let \( I \) be a finite index set, \( \{ P_i \}_{i \in I} \) a set of security policies, and \( x \in (\bigcup_{i \in I} \text{Dom}_{P_i})^k \) a \( k \)-dimensional vector of subjects and objects. The classification function \( c^k \) is defined for \( x \) by

\[
\delta^k(x) = \begin{cases} 
P_j & \forall 1 \leq l \leq k : |\Pi_{x_l}| = 1 \land \Pi_{x_l} = \{P_j\} \quad \text{(class-one interaction)} \\
T & |\bigcap_{1 \leq l \leq k} \Pi_{x_l}| = 0 \quad \text{(class-two interaction)} \\
F & \big|\bigcap_{1 \leq l \leq k} \Pi_{x_l}\big| \geq 1 \land \exists x_l, 1 \leq l \leq k : |\Pi_{x_l}| > 1 \quad \text{(class-three interaction).} 
\end{cases}
\]

For \( k = 2 \), the definition of \( c \) in section 3 is a special case of this more general version.

As an example, consider two policies \( P \) and \( Q \) with \( S_P = \{s_{p_1}, \ldots, s_{p_k}, s_x\} \), \( O_P = \{o_{p_1}, \ldots, o_{p_l}, o_x\} \), \( S_Q = \{s_{q_1}, \ldots, s_{q_m}, s_x\} \) and \( O_Q = \{o_{q_1}, \ldots, o_{q_n}, o_x\} \). Figure 5 shows the access matrix of a corresponding policy group \( M_G \) constructed according to the definition.

![Access Matrix](image)

Note that the definition provides a detailed construction plan for the HRU model of a policy group. This plan can in practice be used to construct the model automatically, once all the HRU models of the \( P_i \), \( T \) and \( F \) exist.

In general, the transition function of any security policy model is no partial function; a policy is defined for any security-relevant operation within its domain. However, \( \text{Dom}_T = \text{Dom}_F = \bigcup_{i \in I} \text{Dom}_{P_i} \) implies that \( \delta_T \) is also defined for any class-one and class-three access which is not its designated role. The same argument holds for \( \delta_P \) for any class-one and class-two access. While the construction of \( \delta_G \) in the above definition actually prevents \( F \) and \( T \) to be applied to the "wrong" access class and thus guards the autonomy of the \( P_i \), a discussion of the safety properties of policy groups becomes easier if we put a corresponding constraint directly on \( T \) and \( F \). So, without loss of generality we define the term of a \( P_1 \)-autonomous policy group.
Definition ($P_i$-Autonomy)

A HRU-model $(Q, E, \delta, q^0)$ with $Q = (S, O, M)$ is called constrained with respect to a set of entities $X \subseteq S \cup O$ if

(a) the domain of $\delta$ is constrained to entities in $X$, in other words, $E = A \times X^k$ holds

(b) the range of $\delta$ is constrained to $Q = (S|_X, O|_X, M|_X)$, where $M|_X$ means the constraint of $M$ to such cells $(s, o)$ for which $s$ and $o$ are in $X$.

A HRU-model of a policy group $G = ([P_i])_{i \in I}, T, F, c)$ is called $P_i$-autonomous if $T$ is constrained to $\{x | x \in (S_G \cup O_G)^k \land \delta^k(x) = T\}$ and $F$ is constrained to $\{x | x \in (S_G \cup O_G)^k \land \delta^k(x) = F\}$.

$P_i$-autonomy guarantees that the completeness and conflict policies do not interfere with any $P_i$ for those access operations that $P_i$ can handle autonomously. The following two examples will illustrate these definitions. Consider again that Ann, employee in department $Q$, has a temporary assignment to department $C$. At the same time, Joe from department $C$ has an assignment to department $Q$ (Figure 6).

![Figure 6: A two-policy scenario with interdomain accesses](image)

All accesses between the principals of this scenario belong to access classes one and two: $(Ann, Joe's file)$ and $(Joe, Ann's file)$ belong to class one, and $(Ann, Ann's file)$ and $(Joe, Joe's file)$ belong to class two. The policy group for policies $C$ and $Q$ is $([C, Q], T, F, c)$, its HRU model is $([\{Joe, Ann\}, \{Joe's file, Ann's file\}, M], E, \delta, q^0)$. Figure 7 shows $M$, where the cells show the policy that controls the corresponding access.

<table>
<thead>
<tr>
<th></th>
<th>Joe's file</th>
<th>Ann's file</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T$</td>
<td>$Q$</td>
</tr>
<tr>
<td>Ann</td>
<td>$C$</td>
<td>$T$</td>
</tr>
</tbody>
</table>

![Figure 7: The access matrix for the two-policy scenario](image)

In our second example we remember that $Q$'s domain is embedded in the domain of the $R&D$ branch (Figure 8). Within the $R&D$ domain, another employee, Lee, works at his regular place. Whenever either Joe or Ann's file is involved, we now additionally encounter possible conflicts between $R&D$ and $Q$. Figure 9 shows the access matrix of the corresponding policy group model.
Let us now turn to the safety properties of policy groups. Our first statement is about the general decidability of the safety problem for policy groups.

**Statement (Decidability of the safety problem for policy groups)**

The general problem of determining whether a given state is safe for a given right in a HRU-modelled policy group is undecidable.

This statement is quite obvious, because by our definition any HRU model of a policy group is at least as expressive as the general HRU model which is known to be expressive enough to simulate any given Turing machine. The statement then follows immediately from the results of [8].

The following statement shows a simple condition based on the properties of the $P_i$ that is already sufficient to make the safety problem for policy groups undecidable.

**Statement**

Let all $P_i$ be HRU-modelled security policies and $G$ a $P_i$-autonomous policy group for the $P_i$. If there exists at least one $j \in I$ where $Dom_{P_i} \cap Dom_{P_j} = \emptyset$ for all $i, j \in I, i \neq j$, and the safety problem is undecidable for $P_j$, then the safety problem is undecidable for $G$.

**Proof**

The proof lifts the local undecidability within $P_j$ to $G$.

Firstly, let purge be a filtering function that removes all operations from a sequence $e$ of inputs that cannot be handled by a given policy $P$ alone:

$$\text{purge}(e, P) = e$$
Secondly, let \( q \) be some state of policy \( P_j \). Then, for any \( e \) being an arbitrary input sequence with elements from \( E_G \) and \( \delta^* \) as defined in the appendix, we define

\[
(S_j, O_j, M_j) := \delta_j^*(q, \text{purge}(e, P_j))
\]

\[
(S_G, \hat{O}_G, M_G) := \delta_G^*(q, e).
\]

Because \( \text{Dom}_{P_j} \cap \text{Dom}_{P_i} = \emptyset \) for any \( i, j \in I, i \neq j \), \( |\Pi_{x_l}| = 1 \cap \Pi_{x_l} = \{P_j\} \) holds for any \( 1 \leq l \leq k \) and any input \( e = (a, x) \) with \( x \in (\text{Dom}_{P_j})^k \). In other words, \( \delta^*(x) = P_j \) holds. Because \( G \) is \( P_i \)-autonomous, \( M_{G, \text{Dom}_{P_i}} = M_j \) holds. Thus any input sequence for \( G \) results in a matrix \( M_G \) that, constrained to the domain of \( P_j \) is identical to the matrix of \( P_j \) after the same (purged) sequence has been executed by \( P_j \). In other words, \( G \) behaves locally in \( P_j \) exactly like \( P_j \) would behave alone; the non-decidability within \( G \) now follows from the non-decidability within \( P_j \). \(
\]

Note that the statement "If the safety problem is undecidable for each \( P_i \) then it is undecidable for \( G \)" is not true in general. A simple example demonstrating the opposite is two policies \( P_1 \) and \( P_2 \) with \( \text{Dom}_{P_1} = \text{Dom}_{P_2} \); here, each access causes a conflict, and neither \( P_1 \) nor \( P_2 \) are ever used. Then, for the safety problem to be decidable in \( G \) it is sufficient that the safety problem is decidable in \( F \).

**Statement (on monooperational \( P_i \))**

Let \( G = (\{P_i\}_{i \in I}, T, F, c) \) be a policy group for policies \( P_i \) and \( (Q_G, E_G, \delta_G, q_0) \) be its HRU-model. If each \( P_i \), \( T \) and \( F \) are monooperational, then there is an algorithm that decides if \( G \) is safe for a given state and a given right.

**Proof**

Our definition of \( G \) composes \( \delta_G \) from the \( \delta_i \)'s, \( \delta_T \) and \( \delta_F \). These building blocks themselves are monooperational compositions of HRU-primitives. Thus \( \delta_G \) consists only of monooperational compositions of HRU-primitives, and \( G \) is monooperational. The statement then follows directly from the theorems in [8].

**Statement (on static \( P_i \))**

Let \( G = (\{P_i\}_{i \in I}, T, F, c) \) be a policy group for policies \( P_i \) and \( (Q_G, E_G, \delta_G, q_0) \) be its HRU-model. If each \( P_i \), \( T \) and \( F \) are static, then there is an algorithm that decides if \( G \) is safe for a given state and a given right.

The proof goes along the same lines as the proof for monooperational models.

**Statement (on monotonic and monoconditional \( P_i \))**

Let \( G = (\{P_i\}_{i \in I}, T, F, c) \) be a policy group for policies \( P_i \) and \( (Q_G, E_G, \delta_G, q_0) \) be its HRU-model. If each \( P_i \), \( T \) and \( F \) are monotonic and monoconditional, then there is an algorithm that decides if \( G \) is safe for a given state and a given right.

Again the proof goes along the same lines as the proof for monooperational models.
5 Implementation

We are implementing policy groups as part of the security policy implementation within the Distributed Computing Environment (DCE) of the Open Systems Foundation [5]. The conflict and completeness policies $T$ and $F$ are implemented in exactly the same paradigms and are enforced with exactly the same mechanisms as any regular security policy within the DCE.

The classification function $c$ is part of the system’s reference monitor that implements the total access mediation property. It substitutes the regular security policy call that is issued by the reference monitor on every entity interaction. While class-one interactions proceed to the responsible security policy, class-two and class-three interactions are diverted to $T$ respectively.

Any component of a reference monitor that is involved in implementing the total access mediation property is crucial to a system’s overall performance. The computational complexity of $c$ depends on the complexity of determining the membership of an entity to all the system’s policy domains and on the complexity of set primitives such as computing set intersection or the cardinality of a set. Considering that while entity interactions occur frequently, security domains in general change rather rarely, our decision was to aim at an implementation with fast lookups and slow updates. As a consequence, updates of a policy domain run in linear time depending on the number of entities joining or leaving a domain; classifying a particular interaction runs in constant time. A very simple implementation for the OSF DCE can be found at http://set.gmd.de/~kuhnhaer/mp.ps; this implementation omits reference counters and hash-keyed set implementation but still updates a policy domain in $O(|I| \times |\bigcup_{i \in I} \text{Dom}_P|)$.

6 Conclusions

We described an approach to secure interdomain operations in multipolicy systems. We identified two major problem areas: interdomain actions that cause conflicts between the involved security policies and interdomain actions for which none of the involved policies provides any security rule.

The paper argued that in order to deal with these obstacles, new security requirements for interdomain actions must be furnished and implemented by additional interdomain security policies. To this end, the notion of policy groups was introduced. Policy groups compose a set of regular security policies and a set of interdomain security policies into a single structure, hereby providing a single point of reference for the discussion and analysis of a multipolicy system’s security properties.

The security-critical nature of interdomain actions implies a major concern for the analysis of a policy group’s security properties. To this end, the paper introduced a formal model based on Harrison/Ruzzo/Ullman’s access control calculus. The model includes a detailed construction plan that in practice can be used for the automatic composition of a policy group’s model.

The paper concluded with statements about the decidability of the safety problem for policy groups and about the computational complexity of an implementation.
Appendix

Definition (HRU-model of a security policy)

A HRU-model of a security policy is a state machine \((Q,E,\delta,q^0)\) where

\[ Q = (S,O,M) \]

is the set of states with

\( S \) is the set of subjects,

\( O \) is the set of objects,

\( M : S \times O \to \wp(R) \) is an access matrix over a finite set of rights \( R \),

\( E = A \times X \) is the set of inputs where \( A \) is the set of access operations

and \( X = (S \cup O)^k \),

\( \delta : Q \times E \to Q \) is the transition function, and

\( q^0 \) is the initial state.

Each input models the call of an application-specific operation \( a \in A \) that involves \( k \) subjects and objects. \( \delta \) models the access-control-related semantics of such calls by statements in the general form

\[
\delta(q,(a,x)) : = \text{IF } r_1 \in M(x_{s1},x_{o1}) \text{ AND } ... \text{ THEN } p_1;...;p_n \text{ FI},
\]

where \( q \) is a state in \( Q \), \( a \) an application operation in \( A \), \( x \) a \( k \)-dimensional vector of subjects and objects in \( S \cup O \), \( r_i \) a right in \( R \), and the \( p_i \) are from the set \( H \) of HRU primitives

\[
H = \{ \text{enter } r \text{ into } M(x_s,x_o), \text{ delete } r \text{ from } M(x_s,x_o), \text{ create subject } x_s, \text{ create object } x_o, \text{ destroy subject } x_s, \text{ destroy object } x_o \}. \]

The safety of any state \( q \) of such a model with respect to a given right takes into account all possible state sequences, beginning with \( q \).

Definition (Safety of a HRU model)

Let \( \alpha \circ e \) be a sequence of inputs from \( E^* \), consisting of a single input \( \alpha \in E \cup \{ \varepsilon \} \) (\( \varepsilon \) is the empty sequence) followed by a sequence \( e \in E^* \). We define \( \delta^* : Q \times E^* \to Q \) by

\[
\delta^* (q,\varepsilon) = q \\
\delta^* (q,\alpha \circ e) = \delta^* (\delta(q,\alpha),e)
\]

A state \( q^0 = (S^0,O^0,M^0) \) of a given HRU-model is called HRU-safe with respect to a right \( r \in R \) if and only if beginning with \( q^0 \) there is no sequence of states that enters \( r \) into some cell of \( M \) that did not already contain it in \( q^0 \). In other words, for all \( s \in S, o \in O \) and all \( e \in E^* \)

\[
r \notin M^0(s,o) \Rightarrow r \notin M^k(s,o) \text{ with } q^k = \delta^* (q^0,e)
\]

holds.
References


