# EFFICIENT ALGORITHMIC SAFETY ANALYSIS OF HRU SECURITY MODELS*

Anja Fischer and Winfried Kühnhauser
*Ilmenau University of Technology, Ilmenau, Germany*

Keywords: Security engineering, Security policies, Security models, Access control, HRU safety, Model decomposition, Enterprise resource planning security.

Abstract: In order to achieve a high degree of security, IT systems with sophisticated security requirements increasingly apply security models for specifying, analyzing and implementing their security policies. While this approach achieves considerable improvements in effectiveness and correctness of a system's security properties, model specification, analysis and implementation are yet quite complex and expensive.
This paper focuses on the efficient algorithmic safety analysis of HRU security models. We present the theory and practical application of a method that decomposes a model into smaller and autonomous sub-models that are more efficient to analyze. A recombination of the results then allows to infer safety properties of the original model. A security model for a real-world enterprise resource planning system demonstrates the approach.

## 1 INTRODUCTION

IT systems with advanced security requirements increasingly apply problem-specific security policies for describing, analyzing and implementing security properties (Bryce et al., 1997; Halfmann and Kühnhauser, 1999; Loscocco and Smalley, 2001; Efstathopoulos and Kohler, 2008). In order to precisely describe security policies, formal security models such as (Harrison et al., 1976; Goguen and Meseguer, 1982; Brewer and Nash, 1989; Sandhu et al., 1996) are applied, allowing for formal analyses of security properties and serving as specifications from which policy implementations are generated (Vimercati et al., 2005).

While security policies and their formal models achieve considerable improvements in effectiveness and correctness of a system's security properties, the costs of this approach often are rather forbidding. The analysis of security models is often pestered by computational complexity, making it difficult to devise algorithms for automated analysis tools.

A fundamental motivation for access control security models is to study the proliferation of access rights. This problem was formalized in HRU access control models in order to find proliferation boundaries and to prove that, for a given security model, these boundaries will never be crossed – a security property known as HRU safety.

Unfortunately, HRU safety turned out to be undecidable for models written in the general HRU calculus, making it difficult to devise algorithms for automated safety analysis tools. As a consequence, several safety-decidable fragments of the HRU calculus emerged (Harrison and Ruzzo, 1978; Lipton and Snyder, 1978; Ammann and Sandhu, 1991; Sandhu, 1992) that bought safety decidability by limiting the expressive power of the calculus. Unfortunately, these fragments now result in another drawback: they often show severe limitations in their power to model the complex policies of larger real-world systems. Consequently, the chances for automated safety analyses of real-world security models are quite low.

In this paper, we propose a method for analyzing the safety properties of *unrestricted* HRU security models. Although the safety problem is still generally pestered by undecidability, the method achieves results in many practical cases. The core of the idea is to decompose a model into smaller sub-models, then analyze these sub-models individually and recombine the results – an approach also applied in the analysis of complex automata (Krohn and Rhodes, 1965). The rationale behind this approach is twofold. Firstly, smaller sub-models can be analyzed more efficiently by heuristic safety analysis algorithms with

*This work is partially supported by Carl-Zeiss-Stiftung.

polynomial or exponential runtimes. Secondly, because sub-models can be analyzed independently the parallelism of multi-core processor architectures can be exploited.

The contributions of this paper are a formal definition of the decomposition method and a formal proof of its correctness. Throughout the paper, excerpts from a real-world enterprise resource planning system security policy illustrate the details.

## 2 RELATED WORK

Fundamental to this paper is the work of Harrison et al. (Harrison et al., 1975; Harrison et al., 1976). The authors present a calculus to formalize access control policies, consisting of access control matrices and state machines. The primary purpose of security models written in the HRU calculus is to obtain statements about the proliferation of access rights, drawn from a reachability analysis of the model's state machine. As was to be expected, the HRU safety problem – the property whether a given right may appear in some future automaton state – turned out to be undecidable for general HRU models, and work focused on finding fragments of the HRU calculus with decidable safety properties. It was proved that the safety problem is decidable, e.g. for mono-operational (Harrison et al., 1976) and mono-conditional (Harrison and Ruzzo, 1978) HRU models. Lipton and Snyder (Lipton and Snyder, 1978) introduced static restrictions on subject creation. Sandhu proposed the typed access matrix model (TAM) (Sandhu, 1992) that augmented the HRU model by a type system.

Kleiner and Newcob (Kleiner and Newcomb, 2006; Kleiner and Newcomb, 2007) present an access control model that improves the ability to deal with the absence of permissions but can still simulate any HRU model. The authors also present different and more rigorous terms of safety properties showing that they can be decided. In (Kleiner and Newcomb, 2007) the safety access temporal logic is introduced, which is able to express a variety of safety properties over access control models and is interpreted over finite runs of the access control system. The model checking problem for the entire logic is undecidable; again, a fragment of the logic was identified for which the problem is decidable. The HRU safety problem can be reduced to this fragment in some cases.

Li et al. (Li et al., 2005) argue that delegation of rights is a useful feature of access control systems while it makes safety and security analyses more important. The authors present a trust management language in the RT family (role-based trust-management languages) which can be encoded in the HRU model but is more application-oriented. The authors define the goals of a security analysis in more general terms such as simple safety, simple availability or mutual exclusion and show that they can be decided.

Our approach to HRU safety analysis differs from these approaches in a fundamental way. Instead of introducing model restrictions our safety analysis technique works on the original, general HRU calculus (and, of course, its derivatives) and aims at a practical method for dealing with the problem's computational complexity. We tackle the problem by decomposing a model into smaller sub-models, analyzing the safety properties of the sub-models and providing a technique to carry the analysis results back to the original model. The basic ideas of this approach originate from the decomposition of automata and go back to (Krohn and Rhodes, 1965).

## 3 APPLICATION SCENARIO

The challenges addressed in this paper are motivated by the complexity of the security requirements in real-world application scenarios. In this section, we dedicate some space to introduce a security-sensitive, real-world application system to motivate a set of application-specific security requirements. Based on these requirements we proceed with composing an informal security policy that then motivates the security model in Section 4. The application scenario, the security policy and its model will then provide a background, an example and a source of arguments throughout the rest of the paper.

We do this in some detail, because the subsequent model decomposition technique in Section 5 injects human application knowledge into the safety analysis algorithms as one means to reduce their runtime, and thus this knowledge should be available to the reader.

### 3.1 Application

The application scenario is a distributed enterprise collaboration system providing services that efficiently extend business processes of cooperating organizations across company boundaries. We especially focus on standard business systems (such as enterprise resource planning (ERP) systems) supporting logistic business processes for cooperative order processing.

One important task in this scenario is the process of availability checking (*available-to-promise, ATP*) which is a typical business process that crosses several company boundaries whenever sub-contractors

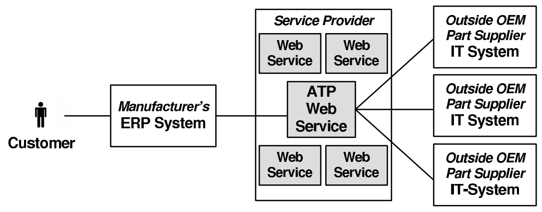

Customer
Manufacturer's ERP System
Service Provider
Web Service
Web Service
ATP Web Service
Web Service
Web Service
Outside OEM Part Supplier IT System
Outside OEM Part Supplier IT System
Outside OEM Part Supplier IT-System

Figure 1: Application scenario.

are involved. When customers place orders with manufacturers that use ERP systems, these ERP systems call ATP services of further suppliers for checking the availability of outside supplied original equipment manufacturer (OEM) parts (Fig. 1). ATP services often are implemented as web services, and thus a manufacturer generally has to call several web services of different OEM suppliers for processing a single order. In our example, this service (among other web services) is provided by a service provider on a shared communication platform. Once the ATP web service has received a manufacturer's availability request, the ATP service forwards the request to all the manufacturer's OEM part suppliers, analyses the responses and sends the results back to the ERP system. The ERP system evaluates the suppliers' responses which then results in an offer for the customer.

## 3.2 Security Requirements

In cross-company business processes, common security properties such as confidentiality, integrity, availability, authenticity, and non-repudiation are both self-evident as well as essential. This section discusses two application-specific key requirements.

One key requirement in this scenario is the *separation of duty* between administrators and regular users. This means, regular users may only obtain regular (operative) permissions and administrators may solely possess administrator permissions.

Delegation of rights — temporarily entrusting a user's access rights to another user — is a common feature of business applications. Due to the collaboration system's properties of providing services to independent companies, the second key requirement is to *restrict delegation of rights* to company boundaries. Consequently, rights may only be delegated between users acting on behalf of the same company.

## 3.3 Authorization Policy

Any security policy is a set of rules designed to meet the security goals of an IT system (Common3.1, 2009). While security policies in general include rules about authentication or communication, the discussion in this paper focuses on the core part of a policy, the authorization rules. Especially, this subsection discusses an informal authorization policy for the example scenario outlined in the previous section. The next section will then rewrite this policy as a formal HRU security model.

The authorization policy is an immanent part of the service provider's security policy and, as such, it is integrated into the service provider's shared communication platform. It contains access control rules for all references to web services[2] made by users. Thus, whenever a user tries to call a web service, the authorization policy's rules are applied to grant or deny access.

In order to enforce separation of duty, the authorization policy differentiates between *regular* and *administrative* rights; a user's right to obtain permissions results from his identity. Among others, the authorization policy contains rights to *execute* a web service and to *delegate* and *revoke* this right. The rights *delegate* and *revoke* realize a user-based grant delegation (Crampton and Khambhammettu, 2008) supporting a one-level delegation hierarchy. Note that the application scenario may also support a $n$-level delegation hierarchy; however, due to demonstrating purposes of the decomposition algorithm we reduce the application scenario to the basics.

In order to implement rights delegation restricted to company boundaries, users from different companies must be distinguished by adopting the concept of security domains of non-interference security models (Goguen and Meseguer, 1982). Hence, to define delegation boundaries, each user is assigned a *security domain*. Users of a company belong to the same security domain, and users belonging to different companies belong to different security domains.

In summary, the authorization policy contains – among others – the following rules:

1. Every user is assigned one of two mutually exclusive user types: regular user or administrator.
2. The user types' rights sets are disjoint. Regular rights are *execute*, *delegate*, and *revoke*. Administrative rights are rights to manage the authorization policy.
3. In order to delegate a right, delegator and delegatee must belong to identical security domains.
4. Delegation of rights is additive — both delegator and delegatee possess the delegated rights.

[2]Note that there are different object types, e.g. web services, administration operations, and rights. Due to conciseness we only consider web services in the following.

## 4 SECURITY MODEL

While security policies generally are phrased in natural language, systems with advanced security requirements increasingly apply security models for formal description and rigorous analysis of their security properties. Most of the existing models address specific challenges: to prove the mapping from an abstract design level (such as information flow graphs) to a system oriented level (such as access control mechanisms (Bell and LaPadula, 1973)), to prove interference rules for security domains (Goguen and Meseguer, 1982), or to precisely define role containment (Sandhu et al., 1996) or information flow (Efstathopoulos and Kohler, 2008).

In large real-world applications, a major challenge is controlling the proliferation of rights. Particularly in distributed systems shared by independent organizations (such as the ERP example), questions such as "Is it possible that users belonging to some organization *A* may, at some point in the future, get access to internal information of users from organization *B*?" arise. The analysis of such dynamic right proliferation is the primary objective of HRU security models and is known as the HRU safety problem (Harrison et al., 1975; Harrison et al., 1976).

In order to model dynamic behavior of access control systems, any HRU security model is a state machine defined by a tuple $(Q, \Sigma, \delta, q^0)$ with a state set $Q$, an input alphabet $\Sigma$, a state transition function $\delta$ and an initial state $q^0$. Any state $q \in Q$ is a snapshot of the system's access control matrix (ACM) and is modeled by a triple $(S, O, m)$ where $S$ is a set of subjects, $O$ is a set of objects, and $m : S \times O \to 2^R$ is an access control matrix where the cells contain subsets of a finite right set $R$.

Security properties can now be analyzed by observing state transitions; in particular, statements about right proliferation can be made by state reachability analysis. Safety analysis in HRU models focuses on a fundamental family of questions: Given some state (an ACM), is it possible that a specific subject ever obtains a specific permission with respect to a specific object? Or, in other words, given a state $q$, is it possible that in some future state $q' = \delta^*(q,a)$ a specific right $r$ leaks into a matrix cell? If this may happen, such states are not considered *safe* with respect to $r$. Precisely, given an HRU security model and a right $r$, a state $q$ is called *safe* wrt. $r$ iff $\forall s \in S, o \in O, a \in \Sigma^*$ :

$$r \notin m(s,o) \Rightarrow r \notin m'(s,o) \text{ with } q' = \delta^*(q,a)$$

(Harrison et al., 1975; Harrison et al., 1976).

Following the policy outlined in Section 3.3, we now will look into the corresponding HRU model; its safety properties will afterwards be analyzed in Section 5.

In our ATP scenario, the set of subjects $S = \{s_i | i \in \mathbb{N}\}$ represents the application's users, whereas the set of objects $O = \{o_j | j \in \mathbb{N}\}$ describes the methods of the web services. $M = \{m | m : S \times O \to 2^R\}$ represents the set of access control matrices. Each matrix $m \in M$ specifies the rights each user (subject) has with respect to each web service method (object).

The set $R$ models rights that subjects may own on objects. In the ERP scenario, users may own rights to execute web service methods (*executeWSMethod*$_r$), to delegate and revoke rights to execute methods (*delegateExecuteRight*$_r$, *revokeExecuteRight*$_r$), or to perform system administration operations such as user, or service management.

Model dynamics are defined by the transition function $\delta : Q \times \Sigma \to Q$ that reflects the rules which prescribe the authorization for making incremental changes to the protection state in the ACM (thus $\delta$ is often called the model's *authorization scheme*). $\Sigma$ is the finite set of inputs covering all application-specific operations that result in a modification of the model state. In real-world applications, operations in $\Sigma$ typically (but not exclusively) are used by security administrators and eventually involve users, web services, and other parameters. For example the operation *createUser* called in some state $q = (S^q, O^q, m^q)$ adds a new subject to $S^q$; here, parameters are the caller (an administrator) and the subject to be created. Precisely, the input set $\Sigma$ is a tuple consisting of

- the set of operations that affect the model state (such as *createUser, delegateExecuteRight*)
- the parameters of these operations, consisting of subjects and objects.

Fig. 2 gives a small example for the definition of a model state transition caused by the operation *delegateExecuteRight*.

```
$\delta(q, (delegateExecuteRight, s_s, s_d, o))$ ::=
  **if** $delegateExecuteRight_r \in m(s_s, o)$
    **and** $executeWSMethod_r \in m(s_s, o)$
    **and** $dom(s_s) \in m(s_d, o_{dom})$
  **then**
    enter $executeWSMethod_r$ into $m(s_d, o)$
  **end if**.
```

Figure 2: Authorization scheme for *delegateExecuteRight*.

$delegateExecuteRight_r$ grants delegator $s_s$ permission to delegate the right $executeWSMethod_r$ to delegatee $s_d$. The rationale is that if $s_s$ owns $executeWSMethod_r$ and is a member of the same security domain as $s_d$, then the state machine moves into a new state in which the access matrix contains the right $executeWSMethod_r$ for the subject $s_d$ with respect to the web service method $o$.

With respect to the pure HRU calculus, we meet with an unfamiliar concept in Fig. 2: we used a function *dom* where an access right should be expected. Although a little off from the core intentions of this section – to sketch an HRU model for an ATP system – we nevertheless want to comment on the validity of doing so.

While from a theoretical point of view the HRU calculus has sufficient expressive power to model any computable policy, from a modeler's point of view the rules of a policy do not always map elegantly to the calculus' abstractions. As an example, while the security policy outlined in Section 3.3 uses security domains in order to isolate different organizations, HRU model states consist of simple ACMs only. We deal with this problem in the following way.

We describe the security domains defined on the policy level by a finite set $D$, and the association of any user to a security domain by a function $dom : S \to D$. Because any function can also be written as a mapping table, we now can express *dom* using the abstractions of the HRU calculus by adding a new column for a virtual object $o_{dom}$ to the access control matrix, in which each domain is represented by a corresponding "right". Thus by extending the set $R$ to $R \cup D$ and the set $O$ to $O \cup \{o_{dom}\}$, a test whether two users $s_s$ and $s_d$ are members of the same domain maps to $dom(s_s) \in m(s_d, o_{dom})$, and the use of the *dom*-function in the condition part of state transition definition becomes legitimate.

Back to modeling the security policy from Section 3.3. Two issues remain to complete the model: modeling the fact that a subject is a security administrator, and the definition of the initial model state. The requirement from Section 3.2 to differentiate between regular users and administrators is met by modeling an authority compartment: admin. We start the model in the initial state $q^0 = (S^0, O^0, m^0)$ with one administrator, resulting in an initial set of subjects $S^0 = \{administrator\}$. Using the same trick as when modeling the *dom* function, the association of subjects to the authority compartment is modeled by a virtual right $admin_r$ which manifests in an extra matrix column for a virtual object $o_{comp}$. Thus the initial object set is $O^0 = \{o_{comp}\}$. The initial matrix $m^0$ consists of a single row for *administrator*, and a single column for the compartment association represented by the virtual object $o_{comp}$ (Fig. 3), and the model is complete.

Concluding, this section has sketched major components of a security model for the policy outlined in Section 3.3 by applying the general HRU calculus, syntactically enriched by calculus-consistent components for modeling administrative users and security domains. The complete model including the definition of all model sets, the complete authorization scheme encompassing about ten administrative operations, and the definition of the initial model state consists of approximately 140 lines in the HRU calculus and took about 16 hours to set up. As a result, we now have a solid foundation for a formal analysis of the security properties and especially the safety properties of our policy.

| $m^0$ | $o_{comp}$ |
|---|---|
| administrator | $admin_r$ |

Figure 3: Initial access matrix $m^0$.

## 5 SAFETY ANALYSIS

In this section, we propose a model decomposition method for unrestricted HRU security models that, although still generally pestered by safety undecidability, allows for safety analysis in many practical cases. The core of the method is to decompose a model into smaller sub-models, then analyze these sub-models individually and recombine the results.

### 5.1 Model Decomposition

Model decomposition divides the state space of an HRU model (the ACM) into two or more smaller and disjoint slices. Each slice differs from the original ACM in that either its subject set, its object set or both are smaller. By maintaining the authorization scheme $\delta$, the input set $\Sigma$ and (accordingly restricted) the initial state $q^0$, two or more sub-models with reduced state spaces emerge.

Just decomposing a model into smaller slices does of course not reduce the inherent complexity of solving the safety problem. The underlying idea is a different one.

Firstly, because the safety problem is generally undecidable, we have to resort to heuristics-based safety analysis algorithms with runtime boundaries typically depending polynomially or exponentially on the size of the model's state space. As an example, for a state space size $s$ and an analysis algorithm running in $O(s^2)$, a state space reduction by a factor of four (model decomposition into four equally-sized sub-models), the runtime of the algorithm is bounded by $4(s/4)^2 = 1/4\, s^2$, which for manageable values of $s$ makes a huge difference to $s^2$. Because sub-models are independent and thus can be analyzed simultaneously on multi-core/multiprocessor/grid architectures, the runtime actually is bounded by $1/16s^2$.

Secondly, the success of heuristic safety analysis algorithms largely depends on the quality of the heuristics, and here we take the view that the use of human problem knowledge is a quality source for guiding heuristics-based algorithms. The challenge here is finding a problem-specific model decomposition that minimizes the state spaces of the sub-models without violating model semantics. On the one hand, a necessary condition for any valid state decomposition is that the resulting slices still are *closed* with respect to the model's authorization scheme $\delta$. On the other hand, a proper heuristics will strive to maximize the number of resulting slices.

Before precisely defining what is meant by *closed with respect to* $\delta$, let us first briefly look at our example from Section 4. For cross-company scenarios, an obvious decomposition heuristics exploits the natural security domains defined by the company perimeters: each sub-model then covers the state of exactly one individual domain. With $D$ denoting the set of domains, this example results in $|D|$ self-contained sub-models with $|D|$ pairwise disjoint subject sets $S_{d_1}...S_{d_{|D|}}$, a common object set $O$, and $|D|$ matrix slices $m_{d_i}$ that contain only the rows for their individual subject sets $S_{d_i}$. Additionally, prominent subjects not directly associated to any security domain (such as security administrators) are collected in a separate slice $m_{d_{adm}}$ (Fig. 4).

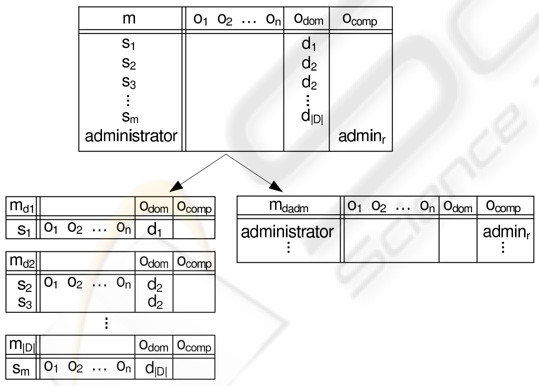


Figure 4: State space decomposition.

Note that in each sub-model only the access matrices differ. The right sets, the rights in the matrix cells and the authorization schemes of the sub-models exactly match their counterparts in the original model.

Now let us consider the properties of a proper model decomposition. Given an HRU model $(Q,\Sigma,\delta,q^0)$, any decomposition heuristics $d$ decomposes the state space $Q = 2^S \times 2^O \times M$ into two or more state spaces $Q_i$ of sub-models $(Q_i,\Sigma,\delta_i,q_i^0)$, where

- $Q_i = 2^{S_i} \times 2^{O_i} \times M_i, S_i \subseteq S, O_i \subseteq O, M_i = \{m|m : S_i \times O_i \rightarrow 2^R\}$
- $\delta_i(q|Q_i,a) = \delta(q,a)|Q_i$ for $q \in Q$ and $a \in \Sigma$
- $q_i^0 = q^0|Q_i$.

The expression "$q|Q_i$" denotes the restriction of a state $q \in Q$ to a smaller state in the state space $Q_i$ of some sub-model. Precisely, $q|Q_i = (S|S_i, O|O_i, m|M_i)$ removes those parts of a state $q$ that are not within the sub-model's definition domain. In the state space decomposition in Fig. 4, a state $q$ of the full model is $(\{s_1...administrator\},\{o_1...o_{comp}\},m)$, and $q|Q_{d_{adm}} = (\{administrator\},\{o_1...o_{comp}\},m_{d_{adm}})$.

Because our goal is to infer safety properties of the original model from individual safety properties of its sub-models, any model decomposition must have distinct properties: each individual sub-model must be an autonomous HRU model, and the set of sub-models collectively must exhibit exactly the same behavior as the original model. We will look into sub-model autonomy and behavior equivalence in the next subsections.

#### 5.1.1 Sub-model Autonomy

In order to be an autonomous HRU model, each sub-model's state space $Q_i$ must be closed with respect to its $\delta_i$. Closedness means that, for each state $q$ of the original model, if $q$ is restricted to the state space $Q_i$ of the sub-model, each state reachable from $q$ in the original model by applying $\delta$ still is in $Q_i$:

**Def. State Closedness.** In any HRU model $(Q,\Sigma,\delta,q^0)$, a subset $Q_i \subset Q$ is called state-closed iff $\forall q \in Q, a \in \Sigma : q \in Q_i \Rightarrow \delta(q,a) \in Q_i$.

#### 5.1.2 Sub-model Isomorphism

In order to infer safety properties of the original model from the set of sub-models, any model decomposition must preserve the structure as well as the behavior of the original model. In other words, a model decomposition $d$ must be an isomorphism.

To this end, for any model decomposition into $n$ sub-models $d$ must induce a *complete* decomposition of the state space $Q$ such that $\bigcup_{i=1}^{n} Q_i = Q$ (denoting that $\bigcup_{i=1}^{n}(S_i \times O_i) = (S \times O)$). Any good decomposition will also avoid redundancies in the sub-models and induce *mutually disjoint* state spaces, $\forall i,j,i \neq j : Q_i \cap Q_j = \emptyset$ (denoting that $(S_i \times O_i) \cap (S_j \times O_j) = \emptyset$).

These properties are summarized in the following definition.

**Def. Model/Sub-model Isomorphism.** Given an HRU model $(Q,\Sigma,\delta,q^0)$, a function

$d : d(Q) \mapsto (Q_1 \dots Q_n)$ is called a decomposition function iff
(a) each $Q_i$ is state-closed
(b) $\bigcup_{i=1}^{n} Q_i = Q$ (completeness)
(c) $\forall i, j, i \neq j : Q_i \cap Q_j = \emptyset$ (mutual disjointness)
(d) $\forall q \in Q, a \in \Sigma : \delta_i(q|Q_i, a) = \delta(q,a)|Q_i$
with $Q_i$ as defined above.

Intuitively, sub-models are autonomous HRU models, sharing the authorization scheme with their origin but having smaller state spaces. The $n$ sub-models generated by $d$ establish a set of $n$ smaller HRU models with state spaces $Q_1 \dots Q_n$, the sub-model set collectively implementing a virtual state transition function $\overline{\delta} : Q_1 \times \dots \times Q_n \times \Sigma \to (Q_1 \dots Q_n)$, $\overline{\delta}(q_1, \dots, q_n, a) \mapsto (\delta_1(q_1, a), \dots, \delta_n(q_n, a))$ with $\delta_i$ as defined in (d). The properties (b) and (c) of the state *space* decomposition function $d$ now imply that there exists a *state* decomposition function $\overline{d} : Q \to Q_1 \times \dots \times Q_n$, $\overline{d}(q) \mapsto (q|Q_1, \dots, q|Q_n)$ that selects the partial states $q_i$ for each sub-model (Fig 5).

Because of (a), (b) and (c), $\overline{d}$ is a homomorphism from $Q$ to $Q_1 \dots Q_n$, and $\overline{d}(\delta(q,a)) = \overline{\delta}(\overline{d}(q), a)$ holds because of (d) (Fig. 6). Because of (b) and (c), $\overline{d}$ is also bijective, and the inverse mapping $\overline{d}^{-1} : Q_1 \times \dots \times Q_n \to Q$ exists which recombines the result of the sub-model's state transitions by $\overline{d}^{-1}(q_1 \dots q_n) \mapsto \bigcup_{i=i}^{n} q_i$. Thus $\overline{d}$ is also an isomorphism, and $\delta(q,a) = \overline{d}^{-1}(\overline{\delta}(\overline{d}(q), a))$ holds, which finally shows that the original model and the set of its sub-models generated by a proper decomposition function $d$ are isomorphic.

$$\begin{tikzcd}
Q \arrow[r, "\overline{d}"] & Q_1 \dots Q_n \arrow[d, "\overline{\delta} = \delta_1 \dots \delta_n"'] \\
 & Q_1 \dots Q_n
\end{tikzcd}$$

Figure 5: State decomposition function $\overline{d}$.

$$\begin{tikzcd}
Q \arrow[r, "\overline{d}"] \arrow[d, "\delta"] & Q_1 \dots Q_n \arrow[d, "\overline{\delta} = \delta_1 \dots \delta_n"'] \\
Q \arrow[r, "\overline{d}"] & Q_1 \dots Q_n
\end{tikzcd}$$

Figure 6: State decomposition isomorphism.

#### 5.1.3 Discussion

The definition of state closedness opens a wide variety of decomposition patterns. In fact, finding a decomposition heuristics that matches the closedness properties is exactly the intellectual challenge that tackles the problem of computational complexity inherent to fully automated safety analysis. In the past, various ACM decompositions such as our own example (Fig. 4), or decompositions into single matrix rows or columns have been successfully applied in different areas. For example, the single row/column decompositions resulted in efficient implementations of ACMs by ACLs respectively capability lists.

We already sketched the results of applying the decomposition method to the security model of our application scenario in Fig. 4. Assuming that the security model consists of $|D|$ security domains and applying a decomposition heuristics based on these domains, we obtain $|D|+1$ sub-models where each sub-model is an autonomous HRU model, and each but one model satisfies the closedness condition.

The rogue sub-model is unique in that it contains only users belonging to the authority compartment $admin_r$ but not to any security domain. At this point, readers already might have observed that this sub-model does not satisfy sub-model closedness, because administrative users are able to affect any other sub-model, e.g. by granting rights to subjects. However, these are desirable characteristics since, being responsible for policy management, administrative users are trustworthy by definition. Furthermore, analyzing a model's safety properties in consideration of administrative users is a trivial problem because they are usually able to proliferate any right in any state and thus are unspectacular with respect to safety analysis. Their collection in a single, unsafe sub-model purges administrative contaminations from all other sub-models and allows for a pure safety analysis restricted to regular users.

## 5.2 From Sub-model Safety to Model Safety

It remains to show that safety properties of an HRU model are maintained by a decomposition. Before we discuss a corresponding theorem, we set up a lemma about the equivalence of $\delta$ and the $\delta_i$'s of the sub-models.

**Lemma ($\delta^*$-equivalence).** Let $\delta^* : Q \times \Sigma^* \to Q$ be the extension of $\delta$ to a sequence of inputs from $\Sigma^*$. Then, given an HRU model $(Q, \Sigma, \delta, q^0)$ and a decomposition function $d$, $\forall a \in \Sigma^*$, $q \in Q$ a sub-model $(Q_i, \Sigma, \delta_i, q_i^0)$ exists such that $\delta^*(q,a)|Q_i = \delta_i^*(q|Q_i, a)$.

This follows directly from the state closedness of the $Q_i$'s and the definition of the $\delta_i$'s, because each $\delta_i$ sees exactly the same local state as its original and thus performs the same state transitions.

**Theorem (Sub-model Safety Equivalence).** Given

an HRU model $(Q,\Sigma,\delta,q^0)$ and a decomposition function $d$. Then, $\forall q \in Q, r \in R$ :
$q$ not safe wrt. $r \Leftrightarrow \exists$ sub-model $(Q_i,\Sigma,\delta_i,q_i^0) : q|Q_i$ not safe wrt. $r$.

**Proof.**
”⇒”:
$q = (S,O,m)$ not safe wrt. $r$
$\Rightarrow \exists\, s \in S, o \in O, a \in \Sigma^*, q' \in Q,$
$q' = \delta^*(q,a) = (S',O',m')$ :
$r \notin m(s,o) \wedge r \in m'(s,o)$.
Because $d$ is complete and disjoint, there exists exactly one sub-model $(Q_i,\Sigma,\delta_i,q_i^0)$ containing the critical matrix cell: $m(s,o) = m_i(s,o)$ where $s \in S_i, o \in O_i$.
Because of the $\delta^*$-equivalence lemma,
$(S',O',m')|Q_i$
$= q'|Q_i = \delta^*(q,a)|Q_i = \delta_i^*(q|Q_i,a) = q_i{}'$
$= (S_i{}',O_i{}',m_i{}')$,
$\Rightarrow m'|M_i = m_i{}'$.
Because $s \in S_i$ and $o \in O_i$
$\Rightarrow m(s,o) = m_i(s,o)$ and $m'(s,o) = m_i{}'(s,o)$.
Because $r \notin m_i(s,o) \wedge r \in m_i{}'(s,o)$, $(Q_i,\Sigma,\delta_i,q_i^0)$ is a sub-model where $q_i = q|Q_i$ is not safe wrt. $r$.
□

”⇐”:
$\exists$ sub-model $(Q_i,\Sigma,\delta_i,q_i^0) : q|Q_i$ not safe wrt. $r$;
then, with $q_i = q|Q_i = (S_i,O_i,m_i)$
$\Rightarrow \exists s \in S_i, o \in O_i, a \in \Sigma^*$:
$q_i{}' = \delta_i^*(q_i,a) = (S_i{}',O_i{}',m_i{}')$ :
$r \notin m_i(s,o) \wedge r \in m_i{}'(s,o)$.
Because of the $\delta^*$-equivalence lemma,
$\forall q \in Q, q_i \in Q_i, q_i = q|Q_i$ :
$(S_i{}',O_i{}',m_i{}')$
$= q_i{}' = \delta_i^*(q_i,a) = \delta^*(q,a)|Q_i = q'|Q_i$
$= (S'|S_i, O'|O_i, m'|m_i)$
$\Rightarrow \forall s \in S_i, o \in O_i : m_i{}'(s,o) = m'(s,o)$.
Because $d$ is a decomposition function, $\forall s \in S_i$, $o \in O_i : m_i(s,o) = m(s,o)$ (property (d)).
Because $r \notin m_i(s,o) = m(s,o) \wedge r \in m_i{}'(s,o) = m'(s,o), q$ is not safe wrt. $r$.
□

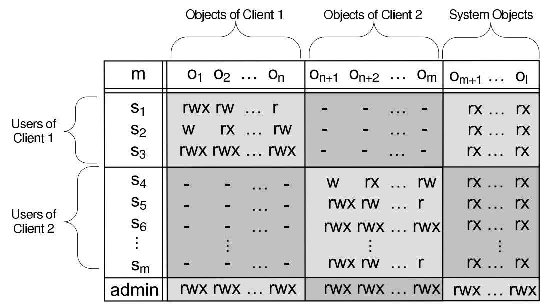
We thus have found precise rules for decomposing any HRU security model into smaller and autonomous sub-models. Because the sub-models are smaller, they generally are easier to analyze and cut down the runtimes of heuristic safety analysis algorithms. Because the sub-models are autonomous, they can be analyzed concurrently.

The properties of the decomposition function guarantee that the original model and its sub-models are structurally equivalent, and that the results of sub-model safety analysis also hold for the original model: on the one hand, for each state that is not safe in some sub-model, a state in the original model exists that is not safe, too; on the other hand, for each state that is not safe in the original model, a state in a sub-model exists that is also not safe.

## 5.3 Applicability of the Decomposition Method

In order to apply the proposed decomposition method, systems need to have a certain property — a system’s HRU model must be decomposable into autonomous sub-models (Section 5.1.2). Just as our application scenario incorporates the required system property due to its security policy rules (separation of duty and restricting right delegations to security domains), numerous state-of-the-art systems also implement this property by design. This section picks two prominent examples from the business and military domain and discusses the applicability of the proposed decomposition method.

In standard business systems, the required system property has been implemented for more than a decade by *multiclient ability* — a concept applied by a variety of standard business software, e.g. by SAP’s business solutions with over 100.000 installations worldwide, e.g. a business software installation can be used by two or more companies in parallel. Multiclient ability allows to map one independent organizational entity to one client on a software installation, without having to install and maintain a separate system for each organizational entity (SAP AG, 2009). Consequently, if a system supports multiple clients, the client’s data – which usually is business-critical – has to be strictly isolated from any other client. This key requirement is usually dealt with by a sophisticated authorization policy — the clients’ users only have rights on the clients’ objects and do not own any rights on objects of the other clients; only users with special privileges (administrators) own rights on all objects.



| | | Objects of Client 1 | | | | Objects of Client 2 | | | | System Objects | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $m$ | $o_1$ | $o_2$ | … | $o_n$ | $o_{n+1}$ | $o_{n+2}$ | … | $o_m$ | $o_{m+1}$ | … | $o_l$ |
| Users of Client 1 | $s_1$ | rwx | rw | … | r | - | - | … | - | rx | … | rx |
| | $s_2$ | w | rx | … | rw | - | - | … | - | rx | … | rx |
| | $s_3$ | rwx | rwx | … | rwx | - | - | … | - | rx | … | rx |
| Users of Client 2 | $s_4$ | - | - | … | - | w | rx | … | rw | rx | … | rx |
| | $s_5$ | - | - | … | - | rwx | rw | … | r | rx | … | rx |
| | $s_6$ | - | - | … | - | rwx | rwx | … | rwx | rx | … | rx |
| | ⋮ | | ⋮ | | | | ⋮ | | | | ⋮ | |
| | $s_m$ | - | - | … | - | rwx | rw | … | r | rx | … | rx |
| | admin | rwx | rwx | … | rwx | rwx | rwx | … | rwx | rwx | … | rwx |

Figure 7: Business software with two clients.

Figure 7 shows the ACM of a multiclient system with two clients. The ACM contains clientdependent objects (business data) for each client and client-independent objects (customizing data) which are system-dependent and shared by all clients. The client's users own the rights to read, write or execute the client's data; the system data can only be read or executed by a client's user. Consequently, this design choice is optimal for the proposed decomposition method since an appropriate decomposition heuristics is directly inferred from the application. As a result, a system hosting $n$ clients can be decomposed in $n+1$ slices (rows), in $n+1$ slices (columns) whereas the client's users and the privileged users are collected in separate slices (see Fig.4), or in $(n+1) \times (n+1)$ slices (checker board pattern).

Military systems commonly enforce *multilevel* security (MLS) policies where objects are classified into hierarchical security levels and subjects are granted clearances meaning that a subject is granted a clearance, only if the subject is considered trustworthy for objects up to this particular security level. The structure of military security levels is commonly described by the lattice model for defining information flow (Denning, 1976). In order to implement these application-oriented MLS policies, system-oriented models based on ACMs have been extended to include security levels, clearances, and classification rules, e.g. the Bell and LaPadula (BLP) model (Bell and LaPadula, 1973) which combines lattices and state machines by defining precise rules (Simple Security Rule, *-property) to control whether or not an ACM is a correct implementation of a lattice. The state transition function thus ensures that the ACM's correctness remains. As such MLS policies modeled with the BLP model are a special case of HRU security models (Pittelli, 1988) which abide to strict MLS policy rules. Both the classification of subjects and objects to a single security level/clearance as well as the strict MLS policy rules set the course for the proposed decomposition method.

Assuming $l$ is an MLS policy's number of security levels assigned to its objects and $c \leq l$ is the number of clearances assigned to its subjects. The system's model can then be decomposed in $l$ slices (columns) where all sub-models have the same subject set but mutually disjoint object sets, or in $c$ slices (rows) where all sub-models have the same object set but mutually disjoint subject sets (Fig. 8 with $c = 3$). Hence, the subjects' clearances respectively the objects' security levels are then disregarded and do not contribute to the decomposition.

In general, MLS policies allow for reclassifying subjects and objects. However, depending on the applied decomposition heuristics reclassifying a subject/object may have an impact on the sub-models'

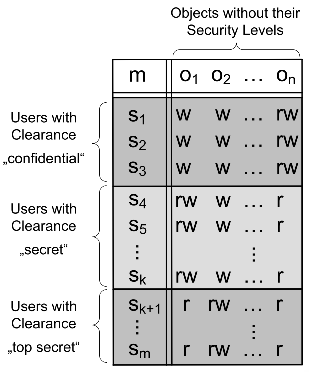


Figure 8: Decomposition according to subject clearances.

subject or object sets. For example, if a model is decomposed according to the subject clearances, reclassifying objects according to policy rules is still state-closed. Granting a subject a new clearance, however, results in removing this subject from the subject set of the old clearance and adding the subject to the new clearance's sub-model. Thus, state-closedness does not hold anymore. On this account, our decomposition model supports reclassification of either subjects or objects. Enabling both requires (i) dynamic subject and object sets within sub-models, and (ii) the decomposition heuristics being part of state transition functions, which is part of our future work.

In summary, a wide variety of state-of-the-art systems can be modeled by the HRU calculus such that the method's requirement of autonomous sub-models is met. Thus, the proposed decomposition method takes a general approach and can be applied in numerous application scenarios.

## 6 CONCLUSIONS

In this paper we described a method for analyzing the safety properties of general, unrestricted HRU security models. The idea is to decompose a model into autonomous sub-models and analyze their safety properties individually. Because the sub-models are smaller, they generally are easier to analyze, and their smaller size significantly reduces the runtime of heuristic safety analysis algorithms with superlinear runtime complexity. Because they are autonomous, sub-models can be analyzed concurrently, allowing to efficiently exploit grid and multiprocessor/multi-core architectures. Strict properties of the decomposition function guarantee that the original model and its sub-models are structurally equivalent, and that sub-model safety properties map to the original model.

Finding a proper model decomposition is a challenge as well as an opportunity: a challenge, because the sub-models have to meet closeness conditions, and an opportunity, because human knowledge is exploited to tackle the computational complexity of the analysis.

While in general safety properties of HRU security models are known to be undecidable, the method allows for safety analysis in many real-world scenarios. Excerpts from the security policy of a real-world enterprise resource planning scenario and a discussion of MLS models support this claim.

## REFERENCES

Ammann, P. E. and Sandhu, R. S. (1991). Safety Analysis for the Extended Schematic Protection Model. In *Proc. IEEE Symposium on Security and Privacy*. IEEE Press.

Bell, D. E. and LaPadula, L. J. (1973). Secure Computer Systems: Mathematical Foundations (Vol.I). Technical Report AD 770 768, MITRE.

Brewer, D. F. and Nash, M. J. (1989). The Chinese Wall Security Policy. In *Proc. IEEE Symposium on Security and Privacy*. IEEE Press.

Bryce, C., Kühnhauser, W. E., Amouroux, R., and Lopéz, M. (1997). CWASAR: A European Infrastructure for Secure Electronic Commerce. *Journal of Computer Security, IOS Press*.

Common3.1 (2009). *Common Criteria for Information Technology Security Evaluation, Version 3.1, Revision 3*.

Crampton, J. and Khambhammettu, H. (2008). Delegation in Role-based Access Control. *Int. Journal of Information Security*.

Denning, D. E. (1976). A Lattice Model of Secure Information Flow. *Communications of the ACM*.

Efstathopoulos, P. and Kohler, E. (2008). Manageable Fine-Grained Information Flow. In *Proc. 2008 EuroSys Conference*. ACM SIGOPS.

Goguen, J. and Meseguer, J. (1982). Security Policies and Security Models. In *Proc. IEEE Symposium on Security and Privacy*. IEEE.

Halfmann, U. and Kühnhauser, W. E. (1999). Embedding Security Policies Into a Distributed Computing Environment. *Operating Systems Review*.

Harrison, M. A. and Ruzzo, W. L. (1978). Monotonic Protection Systems. In DeMillo, R., Dobkin, D., Jones, A., and Lipton, R., editors, *Foundations of Secure Computation*. Academic Press.

Harrison, M. A., Ruzzo, W. L., and Ullman, J. D. (1975). On Protection in Operating Systems. *Operating Systems Review, 5th Symposium on Operating Systems Principles*.

Harrison, M. A., Ruzzo, W. L., and Ullman, J. D. (1976). Protection in Operating Systems. *Communications of the ACM*.

Kleiner, E. and Newcomb, T. (2006). Using CSP to Decide Safety Problems for Access Control Policies. Technical Report RR-06-04, Oxford University Computing Laboratory.

Kleiner, E. and Newcomb, T. (2007). On the Decidability of the Safety Problem for Access Control Policies. *Electronic Notes in Theoretical Computer Science (ENTCS)*.

Krohn, K. and Rhodes, J. (1965). Algebraic Theory of Machines. I. Prime Decomposition Theorem for Finite Semigroups and Machines. *Transactions of the American Mathematical Society*.

Li, N., Mitchell, J. C., and Winsborough, W. H. (2005). Beyond Proof-of-compliance: Security Analysis in Trust Management. *JACM*.

Lipton, R. and Snyder, L. (1978). On Synchronization and Security. In DeMillo, R., Dobkin, D., Jones, A., and Lipton, R., editors, *Foundations of Secure Computation*. Academic Press.

Loscocco, P. A. and Smalley, S. D. (2001). Integrating Flexible Support for Security Policies into the Linux Operating System. In Cole, C., editor, *Proc. 2001 USENIX Ann. Techn. Conference*.

Pittelli, P. A. (1988). The Bell-LaPadula Computer Security Model Represented as a Special Case of the Harrison-Ruzzo-Ullman Model. In *Proc. National Computer Security Conference*. NBS/NCSC.

Sandhu, R. S. (1992). The Typed Access Matrix Model. In *Proc. IEEE Symposium on Security and Privacy*. IEEE.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-Based Access Control Models. *IEEE Computer*.

SAP AG (2009). SAP History. http://www.sap.com/.

Vimercati, S. D. C. d., Samarati, P., and Jajodia, S. (2005). Policies, Models, and Languages for Access Control. In *4th Int. Workshop on Databases in Networkes Information Systems*, Volume 3433/2005 of *LNCS*. Springer.