Beyond Administration: A Modeling Scheme Supporting the Dynamic Analysis of Role-based Access Control Policies

Marius Schlegel[a] and Peter Amthor[b]
Technische Universität Ilmenau, Germany

Keywords: Security Engineering, Security Policies, Access Control, Role-based Access Control Models, RBAC, Heuristic Safety Analysis, Formal Methods.

Abstract: Despite defining a de-facto standard in model-based security engineering, role-based access control models still suffer from limited analysis capabilities. This is especially true for dynamic security properties in the lineage of HRU *safety*. As a consequence, despite of their widespread use for policy specification and implementation, it is difficult to provide and preserve correctness guarantees for such models. We propose a formal framework, called DRBAC, to resolve this dilemma: While retaining application-oriented model abstractions, our approach allows to configure their dynamics in terms of state transitions. This enables a security engineer to tailor both a model and its analysis method to certain safety-related analysis goals. We demonstrate this claim based on a practical security policy.

1 INTRODUCTION

To meet mission-critical security requirements, access control (AC) systems must guarantee to implement a correct security policy. For decades, such correctness guarantees have been founded on formal methods (Vimercati et al., 2005; Tripunitara and Li, 2007; Basin et al., 2011) which rely on three fundamental artifacts: (1.) a formal security *model*, e. g. based on first-order logic or set algebra, (2.) a formal definition of its correctness condition (*security property*), and (3.) a method as automated as possible to validate the former against the latter. Decades of experience and best practices have yielded *modeling schemes*, formal toolboxes of predicates, set-theoretical expressions, automata theory etc., that help engineers in choosing suitable formalisms for a security model.

Role-based access control (RBAC) is a modeling scheme which has been established as a de-facto standard for a wealth of application scenarios (Sandhu et al., 1999; Sandhu et al., 2000). Despite its generalization through attribute-based modeling schemes (ABAC) (Fernández et al., 2019; Chakraborty et al., 2020), it remains one of the most relevant formal tools because of its simple, clear, yet practical abstractions. In contrast to the modeling scheme itselft, however, there are no standardized artifacts to answer the other two questions involved in RBAC policy analysis: how to formally express relevant security properties, and how to analyze them based on automatable methods.

This is especially true for the analysis of dynamic state reachability properties, which can be used to foresee privilege escalation vulnerabilities of a policy, but at the same time is pestered by computational complexity (Tripunitara and Li, 2007). Such formal properties, which are known as *safety* properties for historical reasons, need to be (re-)defined for any application-specific AC model. We call the resulting security property the *analysis goal* of that specific model. Since the application-specific definition of an analysis goal is a burdensome and error-prone task, previous research efforts in the AC community have focused on more specialized analysis questions towards the effect of policy administration, rather than those of more general day-to-day accesses performed by users (*administrative RBAC*, ARBAC) (Li and Tripunitara, 2006; Ranise et al., 2014; Calzavara et al., 2015; Dinh et al., 2017).

This paper addresses this limitation: it proposes DRBAC, a novel modeling scheme for RBAC policies, based on a configurable formal framework. This allows a security engineer to derive an analysis goal from the resulting RBAC model, which is by design compatible with a heuristic analysis approach for the most general (and therefore semi-decidable) class of safety properties: dynamic privilege proliferation in

[a] https://orcid.org/0000-0001-6596-2823
[b] https://orcid.org/0000-0001-7711-4450

Schlegel, M. and Amthor, P.
Beyond Administration: A Modeling Scheme Supporting the Dynamic Analysis of Role-based Access Control Policies.
DOI: 10.5220/0009834304310442
In *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications (ICETE 2020) - SECRYPT*, pages 431-442
ISBN: 978-989-758-446-6
Copyright © 2020 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

the lineage of HRU *safety* (Harrison et al., 1976). With such a modeling scheme, we may eventually ensure that precise correctness guarantees can be given and preserved throughout *model-based security policy engineering*, which denotes the whole process of specifying, analyzing and implementing a security policy.

DRBAC is based on the observation that interesting analysis goals relate to (1.) model dynamics not originating from adminstration and (2.) security properties intractable or even undecidable. The first observation can be confirmed in most practical scenarios involving dynamic user management: While both user creation and session logins can be modeled in classical RBAC, they are neither included in administrative model parts used to represent dynamics in ARBAC, nor, as a consequence, are they subject to related work on analyzing role-based policies (cf. Sec. 2). The second observation is based on the mindset that for the most expressive AC models, possibly undecidable *safety* properties may offer insights into policy design errors w. r. t. potential privilege escalation. As indicated by previous work (Amthor et al., 2013; Amthor and Rabe, 2020), heuristic analysis approaches may provide valuable hints to find such errors, but require an appropriate modeling scheme for infinite search spaces.

Paper Organization. After giving an overview of related work in Sec. 2, we present in Sec. 3 our methodical contributions towards model-based security policy engineering: (1.) a uniform framework to express modeling schemes, applied to the ARBAC97 model family; (2.) the DRBAC modeling scheme, including an examplary model for a hospital information system policy which we retain as a running example. Sec. 4 presents our practical contributions towards model analysis: we (3.) demonstrate the benefits that result from our modeling scheme by deriving practically interesting analysis goals; we (4.) illustrate achievable analysis results based on our example policy and (5.) discuss a model-specific DRBAC safety analysis algorithm. We conclude with Sec. 5.

2 RELATED WORK

This work is based on the foundational RBAC96 model family (Sandhu et al., 1996) and its administrative extension, ARBAC97 (Sandhu et al., 1999). Both allow to choose from a set of submodels (RBAC_1 through RBAC_3) to express typical features of an application-specific security policy, such as role hierarchies or separation-of-duty constraints. While dynamic analysis are out of scope of classical RBAC96 models, ARBAC97 models protection state changes as the results of administrative accesses. Again, the modeling scheme provides submodels for different types of dynamics to meet application-specific analysis goals: dynamic change of roles and a role hierarchy (RRA97), user-role-assignments (URA97), and permission-role-assignments (PRA97).

In (Li and Tripunitara, 2006), a first detailed study of dynamic analysis goals and their tractability (including safety) was performed for the URA97 submodel of ARBAC97. The results of this work have been refined and implemented in the MOHAWK line of analysis tools (Jayaraman et al., 2013; Shahen et al., 2015), which allows safety analyses for this model class. Since then, user-role-reachability via URA97 have by far received most attention in the model analysis community: For both ARBAC97 (Stoller et al., 2007; Jha et al., 2008) and related, more specialized modeling schemes (Stoller et al., 2011; Ranise et al., 2014; Calzavara et al., 2015; Shahen et al., 2015; Dinh et al., 2017), analysis techniques for decidable instances of this problem have been presented. This body of work is considered complementary to ours, since it provides efficient methods and tools for specialized classes of policies and their respective analysis goals, while our motivation is to support a broad range of applications for role-based models. This also includes a more generalized notion of safety, which is not restricted to a specific submodel of ARBAC97 – in fact, we suggest to treat the notion of administration independent from model dynamics. This unlocks role-based models to new dynamic analysis approaches, that may also include heuristic methods to reason about undecidable problem instances (which we demonstrate in Sec. 4).

Over the recent years, there has been an increasing attention to ABAC modeling schemes, capable of generalizing roles as attributes (Fernández et al., 2019; Chakraborty et al., 2020). However, since their analysis goals are mainly static (Fernández et al., 2019), formal methods for defining and reasoning about dynamic security properties still benefit from a simplified, yet standardized calculus. To this end, we expect that our RBAC-related results can be carried over to a commonly accepted ABAC modeling scheme (such as proposed in (Jin et al., 2012a; Jin et al., 2012b)).

Our approach builds on previous work to support holistic model-based security policy engineering through configurable formal methods. These propose an automaton-based approach of modeling AC systems (Kühnhauser and Pölck, 2011; Amthor et al., 2013), which consequently allows to tailor each formal artifact used for specifying, analyzing and implementing a security model to the analysis goal required by its application domain (Amthor, 2016; Amthor, 2017). In this paper, we extend this notion of configurable modeling schemes to RBAC: Based on the observation that existing role-based modeling schemes are selected based on either expressiveness (such as with the different submodels of ARBAC97) or the degree of dynamics needed for a certain analysis goal, we propose to unify both goals in one modeling scheme.

3 MODELING SCHEME

In this section we introduce a modeling scheme that reflects the semantics of traditional RBAC policies well-established in both academia and industry. Adding to this, our scheme aims at satisfying the requirements of a holistic model-based security policy engineering process, in particular the need to analyze and verify application-specific, dynamic security properties.

To this end, we first take a look at this engineering process in order to highlight the origins of application-specific formalization requirements. The mindset of our approach is then to fine-tune all formal artifacts involved in both model engineering and model analysis towards these requirements. We call this idea *model configuration*.

3.1 Engineering Process

For engineering security-critical systems, both specification and verification of security policy rules is based on formal security models. On a fundamental level, the reasons for this are twofold: First, the unambiguous representation of such rules provides a sound foundation for their correct implementation. Due to their application-oriented abstractions, administration-friendliness and resource-efficient enforcement, standardized RBAC models (Sandhu et al., 1996; Sandhu et al., 1999; Sandhu et al., 2000; Ferraiolo et al., 2007) have proven to satisfy this purpose. Second, diverse modeling schemes enable the analysis of application-specific security properties on a formal level. In the last decades, specialized modeling schemes have emerged for certain analysis goals (Harrison et al., 1976; Vimercati et al., 2005; Barker, 2009; Ranise et al., 2014).

The usage of security models has led to model-based security policy engineering (MSPE) as a specialization of the generic software engineering processes. It can be divided into three steps (cf. Fig. 1):

1. *Model Engineering* denotes composing a formal model from the informal representation of a security policy, based on some modeling scheme that supports the subsequent steps. This model is then instantiated, e. g. its primitive components are initialized according to the policy. We hence call the result of model engineering an *instance* of the chosen security model.
2. *Model Analysis* comprises analyzing the model instance against security properties, which involves possible re-iteration of model engineering decisions as supported by the modeling scheme. This leads to possible adjustments in both the previously derived model and its initialized components. The finally resulting model instance is meant to comply to the application-specific analysis goal(s).
3. *Model Implementation* denotes implementing the model instance in software.

As becomes apparent, the merits of formal models are leveraged in different steps, which possibly leads to a practical dilemma: Modeling schemes developed for model engineering, such as RBAC, are usually impractical if not impossible to use when targeting certain analysis goals, such as privilege escalation. Modeling schemes developed for such goals, on the other hand, may fail to represent application-specific policy semantics. This situation leads to models for analysis that are totally different, from a semantical point of view, from models used for specification and, finally, for implementing a security policy. The resulting semantic gap between correctness guarantees gained during model analysis and the model specification used for its implementation quite possibly leads to errors that contradict the raison d'être of the formal approach.

To resolve this dilemma, more flexible modeling schemes are needed that adapt to their usage in both model engineering and model analysis. In previous work (Kühnhauser and Pölck, 2011; Amthor et al., 2014; Pölck, 2014; Amthor, 2016; Amthor, 2017), we propose an approach that allows a policy engineer to configure modeling schemes to application-specific analysis goals. From a practical perspective, typical analysis goals may describe different levels of dynamics w. r. t. the evolution of a system – usually represented by a formalization of *protection states* and *state transition* rules. Combining such goals with RBAC policies leads to a variety of possible RBAC models ranging from completely dynamic (every part of the model definition can change during runtime) to completely static (no changes during runtime allowed). The latter is already covered by RBAC96, while ARBAC97 provides a formalism to express limited dynamics originating from administration; however, for all remaining use cases a uniform modeling scheme is still missing.

In the remainder of this section, we demonstrate the idea of model configuration based on a real-world use case. We introduce a configurable RBAC modeling scheme, which can be fine-tuned to application-specific analysis goals regarding dynamic behavior.

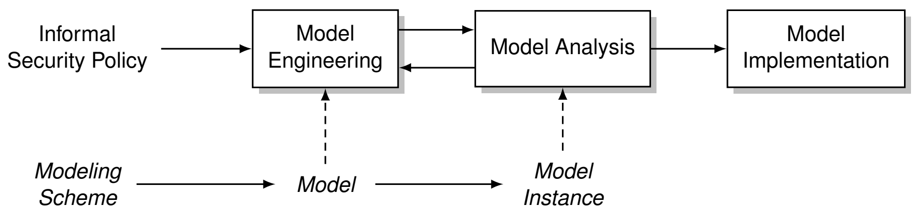

Figure 1: Steps in model-based security policy engineering. Formal artifacts are printed italic, dashed arrows denote "used-in".

As a basis for this, we first introduce an exemplary security policy.

3.2 Examplary Security Policy

As a running example, the rest of this paper uses a security policy for a hospital information system (HIS), simplified for illustration purposes. We will now informally describe this policy.

There are five different **roles** to express disjoint capabilities of doctors, nurses, patients, managers and receptionists. There are also more specialized roles for doctors and nurses working in different hospital wards, including ICU, cardiology and maternity. These roles extend the permissions assigned to their base roles.

Permissions allow viewing, creating or modifying electronic patient records (EPRs) and a service roster, delegating the treatment of a patient to a referred doctor in a different ward and logging in to or out from the system. Any doctor assigned to some ward may both view and modify EPRs in that ward, while nurses assigned to the same ward may only view them. Both doctors and nurses from any ward may view the service roster, which can be both viewed and modified by a hospital manager. Doctors may consult their colleagues on a case by temporarily delegating treatment capabilities in their own ward to another doctor, who may in turn delegate the case to a third doctor etc. Receptionists may create EPRs for new patients. Any user may log in or out.

The above permissions lead to the following user **operations** for the HIS scenario: view EPR, modify EPR, create EPR, view service roster, login, logout, delegate treatment. Modifying the service roster is considered an administrative operation, since it reassigns a doctor or nurse to some ward.[1] Ward-specific roles for doctors or nurses are required for using any particular permissions for the respective ward. Since users may change their ward, their respective base roles are a prerequisite for the adoption of a ward-specific role.

[1]For simplicity, although required from a practical standpoint, we have refrained from modeling operations to delete users and revoke roles.

3.3 Role-based Model Components

As a first step to formalize a security policy, any modeling scheme relies on what we call *primitive components* (or just components). They are the basic building blocks used for two purposes: First, defining elementary identifies, such as for users, roles or permissions; second, for defining their interrelations in terms of policy rules, such as for evaluating access decisions or restricting possible protection state changes. In case of role-based policies, RBAC96[2] defines a minimum of eight primitive components, which can be formally assigned to exactly one of these purposes.

Definition 1 (***Primitive Model Component***)**.** A primitive model component is either (1.) a set $\aleph_i$ of elementary values or (2.) a relation, mapping or boolean expression $\beth_j$ defined on the basis of $\aleph$ components.

We further call these two "$\aleph$ components" and "$\beth$ components", respectively. Typically, $\aleph$ components are used to specify (possibly infinite) sets of elementary identifiers, while $\beth$ components associate them with each other in a meaningful way (e. g. to specify authorization rules).

Example 1. Consider our exemplary HIS policy as introduced in Sec. 3.2. An RBAC model is used to formalize this policy, which contains the components listed in Tab. 1 (columns 1–3).[3] We assume the choice of model components was part of the requirements-driven design of the informal security policy. Then, during model engineering, an instance of the model may be defined as shown in Tab. 1 (column 4).

According to the idea of administration, this model should be extended by ARBAC components to model role assignments. We have listed these components in Tab. 2 (column 1–3). Note that for our policy there is no need for an administrative role hierarchy (*ARH*).

[2]For the rest of this paper we will use the term RBAC to denote the feature-complete RBAC_3 model from RBAC96. In a similar manner, when talking about ARBAC97, we will just refer to ARBAC and URA.

[3]We use "_" as a wildcard to support legibility, which expands to any non-empty element in the respective name context (only used if unambiguous).

Table 1: RBAC primitive model components and their instantiation for Example 1. Column 1 shows the component class.

(1)	(2) Symbol	(3) Description	(4) HIS Instantiation
$\aleph_1$	U	set of user identifiers	{drCox, drKelso, drJD, nurseCarla, nurseLaverne, mrsFriendly, mrBruise, msPregnant}
$\aleph_2$	R	set of role identifiers	{rDoctor, rDoctorICU, rDoctorCard, rDoctorMat, rNurse, rNurseICU, rNurseCard, rNurseMat, rPatient, rReceptionist}
$\aleph_3$	P	set of permission identifiers	{viewRec, createRec, modRec, viewService, delegTreat}
$\aleph_4$	S	set of session identifiers	$\emptyset$
$\beth_1$	$UA \subseteq U \times R$	user-role-relation	$\{\langle \text{drJD}, \text{rDoctor} \rangle, \langle \text{drCox}, \text{rDoctor} \rangle, \langle \text{nurseCarla}, \text{rNurse} \rangle, \langle \text{mrsFriendly}, \text{rReceptionist} \rangle, \langle \text{mrBruise}, \text{rPatient} \rangle, \ldots\}$
$\beth_2$	$PA \subseteq P \times R$	permission-role-relation	$\{\langle \text{viewRec}, \text{rDoctor} \rangle, \langle \text{modRec}, \text{rDoctor} \rangle, \langle \text{viewService}, \text{rDoctor} \rangle, \langle \text{delegTreat}, \text{rDoctor} \rangle, \langle \text{createRec}, \text{rReceptionist} \rangle, \ldots\}$
$\beth_3$	$user : S \to U$	session-user-mapping	(undefined)
$\beth_4$	$roles : S \to \mathcal{P}(R)$	session-roles-mapping	(undefined)
$\beth_5$	$RH \subseteq R \times R$	role hierarchy (partial order)	$\{\langle \text{rDoctorICU}, \text{rDoctor} \rangle, \langle \text{rDoctorCard}, \text{rDoctor} \rangle, \ldots, \langle \text{rNurseICU}, \text{rNurse} \rangle, \langle \text{rNurseCard}, \text{rNurse} \rangle, \ldots\}$
$\beth_6$	*Constraints*	model constraints	$\forall u \in U : (\langle u, \text{rDoctor\_} \rangle \in UA \Rightarrow \langle u, \text{rDoctor} \rangle \in UA) \wedge (\langle u, \text{rNurse\_} \rangle \in UA \Rightarrow \langle u, \text{rNurse} \rangle \in UA)$

Table 2: ARBAC primitive model components and their instantiation for Example 1. Column 1 shows the component class.

(1)	(2) Symbol	(3) Description	(4) HIS Instantiation
$\aleph_5$	AR	set of administrative role identifiers	{rManager}
$\aleph_6$	AP	set of administrative permission identifiers	{modService}
$\beth_7$	$AUA \subseteq U \times AR$	user-administrator-relation	$\{\langle \text{drKelso}, \text{rManager} \rangle\}$
$\beth_8$	$APA \subseteq AP \times AR$	permission-administrator-relation	$\{\langle \text{modService}, \text{rManager} \rangle\}$

Again, these ARBAC components add to our exemplary model instance as shown in Tab. 2 (column 4).

At last, to describe dynamics of the administrative operation that re-assigns doctors or nurses, we define the *can_assign* relation in the URA submodel:

$$\begin{aligned} can\_assign = \{ & \langle \text{rManager}, \text{rDoctor}, \{\text{rDoctor\_}\} \rangle, \\ & \langle \text{rManager}, \text{rNurse}, \{\text{rNurse\_}\} \rangle \}. \end{aligned}$$

This allows any user acting as rManager to assign one of the ward-specific roles _ICU, _Card, _Mat to any user with the base role rDoctor or rNurse.

This example already exposes one limitation of the traditional modeling schemes: While RBAC model components represent our intended policy semantics quite well, they do not allow reasoning about dynamic model properties such as the future evolution of role assignments and role activations. ARBAC, on the other hand, only allows us to model a small subset of possible state changes: Using the *can_assign* relation, we may restrict role re-assignments through a certain type of pre-condition. However, as we can see from the example, richer semantics for both preconditions as well as dynamic protection state changes are required: These include user operations to login (activate roles), create new users (new patient EPRs) or delegate permissions to another user. All of these cannot be modeled without bending the intended semantics of administrative roles and permissions in the above ARBAC model.

3.4 DRBAC Modeling Scheme

In the following we discuss a more flexible and analysis-friendly RBAC modeling scheme, which allows reasoning about dynamic model properties. To this end, we treat the AC system as a deterministic automaton, based on the original idea of (Harrison et al., 1976). The idea of our dynamic RBAC modeling scheme (DRBAC) is to combine both primitive model components established for traditional RBAC models (see Def. 1 and Example 1) with an automaton-based modeling scheme that generalizes dynamic behavior.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Definition 2 (*Dynamic AC Model*). A dynamic access control model is a deterministic automaton defined by a tuple $\langle \Gamma, \Sigma, \Delta \rangle$, where

- the *state space* Γ is a set of protection states;
- the *input set* $\Sigma = \Sigma_C \times \Sigma_Y^*$ defines possible inputs that may trigger state transitions, where Σ_C is a set of *command* identifiers used to represent operations a policy may authorize and Σ_Y is a set of *values* that may be used as actual parameters of commands;[4]
- the *state transition scheme (STS)* $\Delta \subseteq \Sigma_C \times \Sigma_X^* \times \Phi \times \Phi$ defines state transition pre- and post-conditions for any input of a command and formal parameters, where Σ_X denotes a set of *variables* to identify such parameters.

We use Φ to represent the set of boolean expressions in first-order logic (language-agnostic).

For each $\langle c, x, \phi, \phi' \rangle \in \Delta$, a notation borrowed from the HRU model is used: $c(x) ::= \mathsf{PRE} : \phi; \mathsf{POST} : \phi'$. We call the boolean term ϕ the pre-condition (abbreviated $c.\mathsf{PRE}$) and ϕ' the post-condition (abbreviated $c.\mathsf{POST}$) of any state transition to be authorized via command c. On an automaton level, this means that $c.\mathsf{PRE}$ restricts which states γ to legally transition from, while $c.\mathsf{POST}$ defines any differences between γ and the state γ' reachable by any input word $\langle c, x \rangle$. Since our goal is to reason about possible state transitions, we require that only such commands are modeled in Δ that modify their successor state. To distinguish between the value domains of individual variables in x, we use a refined definition of Σ_X to reflect distinct namespaces of variable identifiers for each model component. These are denoted by sets $X_{\aleph_i}$ so that $\bigcup_{1 \le i \le n} X_{\aleph_i} = \Sigma_X$ for a model with n $\aleph$ components.

One may observe that we do not define an output function for the automaton above. Again, as we are only interested in the effects of state transitions, not access decisions, this is not a necessary part of the modeling scheme. However, for policy-internal authorization rules, we assume a complementary specification is used such as a generic *access control function* (ACF). For example, for RBAC model components $acf_{\mathrm{RBAC}} : S \times P \to \{\mathrm{true}, \mathrm{false}\}$ allows to express if a permission may be used in a certain session.

Both Def. 1 and 2 describe a configurable modeling scheme, capable of modeling any type of protection state change. We again demonstrate its application for our example scenario.

Example 2. For our exemplary HIS policy, a dynamic AC model is defined as follows:

$\Gamma = \mathcal{P}(U) \times \mathcal{P}(S) \times \mathcal{P}(UA) \times USER \times ROLES$, where $\gamma = \langle U_\gamma, S_\gamma, UA_\gamma, user_\gamma, roles_\gamma \rangle \in \Gamma$ is a single protection state,[5]

$\Sigma_C = \{$createPatient, login, logout, assignDoctor, assignNurse, delegateTreatment$\}$,

$\Sigma_Y = U \cup S \cup R$,

$\Sigma_X = X_U \cup X_S \cup X_R$,

Δ is defined by a set of commands as illustrated by four representative examples in Fig. 2. Their semantics are as follows:

createPatient is intended for creating a new user eligible for the role rPatient. As per *PA*, this should be callable by any user logged in as rReceptionist, which is the PRE part of the definition. In POST, this user is created but not already logged in (as this is subject to the *login* command). We assume hospital personnel to be created by similar commands, which assign the base roles rDoctor or rNurse instead of rPatient.

login is the login command for any specific role. In this policy, the base roles rDoctor and rNurse should only serve as assignment prerequisites, i. e. preconditions for modifying *UA* in terms of the model. This is why *login*.PRE checks *UA* before activating the intended role for the calling user in a fresh session. Note that, since our policy explicitly states this, a mere call of the login operation is unrestricted which is why we do not need to check acf_{RBAC} in PRE.

assignDoctor is one of two commands used to assign (“unlock”) a specific role to a user. That user may then use *login* to activate her role. As discussed in the infomal policy, *assign_* commands may only be called by users acting in a privileged role (rManager) allowed to use the modService permission. As an additional term in PRE, any user assigned a specific doctor- or nurse-role must be qualified as a doctor or nurse, respectively – which is modeled by the unassignable, irrevocable base roles rDoctor and rNurse.[6]

delegateTreatment allows any user logged in as a ward-specific *rDoctor_* to non-permanently delegate this role to another doctor. Since the latter is only guaranteed if POST leaves *UA* untouched, we activate the delegated role for a specific, running session of the target user. This specification allows the delegation to be performed transitively, which is intended by our HIS application scenario.

[4]We use the Kleene operator to indicate that multiple parameters may be passed.

[5]Note that we define auxiliary base sets for all possible instances of the mappings *user* and *roles*, such as $USER = \{user_\gamma : S_\gamma \to U_\gamma \mid S_\gamma \subseteq S, U_\gamma \subseteq U\}$ etc.

[6]Technically, this is part of our model constraints which should be checked as a mandatory pre-condition of every command. However, since constraints are static in this example scenario, we have integrated them in *assignDoctor*.PRE, being the only command that may violate them.

▶ **createPatient**$(s_{caller}, u_{patient})$ ::=
PRE: $acf_{\text{RBAC}}(s_{caller}, \text{createRec})$;
POST: $U_{\gamma'} = U_\gamma \cup \{u_{patient}\}$ (a)
$\wedge$ $UA_{\gamma'} = UA_\gamma \cup \{\langle u_{patient}, \text{rPatient}\rangle\}$

▶ **login**$(u_{caller}, r_{activate})$ ::=
PRE: $\langle u_{caller}, r_{activate}\rangle \in UA_\gamma$;
POST: $S_{\gamma'} = S_\gamma \cup \{s^*\}$
$\wedge$ $user_{\gamma'} = user_\gamma[s^* \mapsto u_{caller}]$ (b)
$\wedge$ $roles_{\gamma'} = roles_\gamma[s^* \mapsto \{r_{activate}\}]$

▶ **assignDoctor**$(s_{caller}, u_{doctor}, r_{ward})$ ::=
PRE: $acf_{\text{RBAC}}(s_{caller}, \text{modService})$
$\wedge$ $\langle u_{doctor}, \text{rDoctor}\rangle \in UA_\gamma$
$\wedge$ $r_{ward} \in \{\text{rDoctorICU, rDoctorCard, rDoctorMat}\}$; (c)
POST: $UA_{\gamma'} = UA_\gamma \cup \{\langle u_{doctor}, r_{ward}\rangle\}$

▶ **delegateTreatment**$(s_{caller}, s_{deleg}, r_{ward})$ ::=
PRE: $acf_{\text{RBAC}}(s_{caller}, \text{delegTreat})$
$\wedge$ $r_{ward} \in roles_\gamma(s_{caller})$
$\wedge$ $u_{deleg} = user_\gamma(s_{deleg})$ (d)
$\wedge$ $\langle u_{deleg}, \text{rDoctor}\rangle \in UA_\gamma$;
POST: $roles_{\gamma'} = roles_\gamma[s_{deleg} \mapsto roles_\gamma(s_{deleg}) \cup \{r_{ward}\}]$

Figure 2: Exemplary command definitions for the HIS policy. All non-parameter variables are exists-bound to their respective $\aleph$ component. We use a superscript asterisk $(*)$ to denote a fresh identifier created by the AC system.

Note that, except for the *assign_* commands, each of these commands models a non-administrative access: creating new patients, logging in (or out) and delegating permissions are actions that users should perform as part of their day-to-day workflow rather than a policy administrator. *assign_* on the other hand is a typical example for an administrative operation that needs to be authorized just as a non-administrative access, which involves PRE checking acf_{RBAC}.

When compared to the ARBAC model for this policy introduced in Example 1, we may also observe that the administrative operation modeled be the *assign_* commands precisely matches the pre-conditions earlier defined through the *can_assign* relation. Even more, it also allows for more fine-grained control over both PRE and POST, which enables a customized policy design. This exemplifies how administrative model components present in ARBAC are supported by DRBAC without sacrificing expressive power; at the same time, it eliminates the need to separately define administration-related model components $\aleph_{5\ldots6}$ and $\beth_{7\ldots8}$.

Usage in Model Engineering. Based on the automaton paradigm from Def. 2, we are able to engineer any DRBAC model instance similar to RBAC, by fine-tuning (1.) the expressiveness of the model and (2.) its degree of dynamics. While the first goal is achieved by selecting the required primitive model components from $\aleph_{1\ldots4}$ and $\beth_{1\ldots6}$, the second goal is achieved by configuring the automaton components Γ, Σ and Δ. Beyond our HIS policy in Example 2, this process can be generalized for any set of primitive model components. We therefore conclude this section by demonstrating how submodels of RBAC96 can be defined in a unified manner, using DRBAC.

For this, we introduce the notion of *potential sets* as an auxiliary formalism.

Definition 3 (*Potential Set*). The potential set $\mathcal{P}_\infty(\beth)$ of a primitive model component $\beth$ is defined as follows:

1. if $\beth$ identifies a relation $\times_{i\in\mathbb{N}} A_i$, then $\mathcal{P}_\infty(\beth) = \mathcal{P}(\times_{i\in\mathbb{N}} A_i)$;
2. if $\beth$ identifies a mapping $f : A \to B$, then $\mathcal{P}_\infty(\beth) = \{f : A' \to B' \,|\, A' \in \mathcal{P}(A), B' \in \mathcal{P}(B)\}$ is a set of all possible mappings whose domains and codomains are subsets of A and B, respectively;
3. if $\beth$ identifies a boolean expression, $\mathcal{P}_\infty(\beth) = \Phi$.

The idea of the potential set is to provide a (possibly infinite) range of values for the state-specific instances of dynamic model components. We now apply this idea to concisely define some configurations of the classical RBAC model semantics.

Example 3. A completely dynamic DRBAC model for all RBAC model components is a tuple $\langle\Gamma, \Sigma, \Delta\rangle$ where

$\Gamma = \times_{i=1}^{4} \mathcal{P}(\aleph_i) \times \times_{i=1}^{6} \mathcal{P}_\infty(\beth_i)$;

$\Sigma = \Sigma_C \times \Sigma_Y^*$, where Σ_C is a set of application-specific command identifiers and $\Sigma_Y = \bigcup_{i=1}^{4} \aleph_i$ is a set of argument values usable for these commands;

$\Delta \subseteq \Sigma_C \times \Sigma_X^* \times \Phi \times \Phi$, where $\Sigma_X = \bigcup_{i=1}^{4} X_{\aleph_i}$.

Example 4. A DRBAC model that matches ARBAC in both expressiveness and degree of dynamics is a tuple $\langle\Gamma, \Sigma, \Delta\rangle$ where

$\Gamma = \mathcal{P}(\aleph_2) \times \times_i \mathcal{P}_\infty(\beth_i)$ for $i = 1$ (URA), or $i = 2$ (PRA) or $i = 5$ (RRA);

$\Sigma = \Sigma_C \times \Sigma_Y^*$, where $\Sigma_C = \{$can_assign, can_revoke, can_assignp, can_revokep, can_assigna, can_revokea, can_assigng, can_revokeg, can_modify$\}$ (administrative commands) and $\Sigma_Y = \aleph_2$ (role parameter values);

$\Delta \subseteq \Sigma_C \times \Sigma_X^* \times \Phi \times \Phi$, where $\Sigma_X = X_{\aleph_2}$ (role parameter variables).

Note that, according to the motivation of DRBAC, Example 4 only considers the dynamic model components, while the static components $\aleph_{\{1,3,4\}}$ and $\beth_{\{3,4,6\}}$

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

are still part of the policy. For ARBAC, this is worth mentioning especially for the model constraints ($\sqsupset_6$), which must be checked in PRE for all commands (as discussed in (Sandhu et al., 1996)).

As these examples illustrate, DRBAC offers a wide range of model configuration options in both expressiveness and degree of dynamics while retaining a structurally similar model engineering process, based on a fixed set of once formalized primitive model components. This model engineering approach should be embedded in the MSPE process by tailoring these configuration options to a specific analysis goal, which we will demonstrate in the next section.

4 MODEL ANALYSIS

In the domain of AC models, a core objective is to study the potential proliferation of privileges. First formalized as the *safety* property (Harrison et al., 1976), the following type of questions is addressed: Given a particular model protection state, is it possible that some subject ever obtains a specific privilege with respect to some object? Thus, safety analyses expose model properties that allow for the prediction of whether the dynamic changes of an AC system may lead to an unwanted protection state.

In the previous section, we have presented DRBAC – a modeling scheme for RBAC policies enabling configurable protection state dynamics based on a deterministic automaton. This section aims to carry that approach further to the MSPE step of model analysis by developing a safety analysis method for DRBAC models that can be configured to application-specific safety analysis goals.

First, we explore a family of safety properties for DRBAC models (Sec. 4.1) which are tailorable to application-specific analysis goals. Subsequently, we present an algorithmic framework for DRBAC model safety analyses (Sec. 4.2).

4.1 DRBAC Safety Properties

When analyzing dynamic model behavior regarding safety, we are interested in the effects the dynamic changes will have on future authorizations. Whether a given access request is allowed or denied in the current protection state, is answered by a policy's interface. Similar to a model formalizing a policy's authorization rules, a policy's interface is formalized by its *access control function* (ACF).

In case of the completely static RBAC model, the ACF can be defined as follows (based on (Ferraiolo et al., 2007)).

Definition 4 (*RBAC Access Control Function*). The ACF of an RBAC model $\langle U, R, P, S, UA, PA, user, roles, RH, Constraints\rangle$ is a mapping $acf_{\text{RBAC}} : S \times P \to \{\text{true, false}\}$ where

$$acf_{\text{RBAC}}(s,p) \mapsto \begin{cases} \text{true}, & \exists r, r' \in R, s \in S : \\ & \langle r, r'\rangle \in RH \wedge \\ & r \in roles(s) \wedge \\ & \langle p, r'\rangle \in PA, \\ \text{false}, & \text{else.} \end{cases}$$

The semantics of acf_{RBAC} is that the request of a session s for a permission p (e. g. performing an operation view on an EPR object) is allowed iff a role r is activated for s, for which p is assigned to r or a role r' subordinated to r.[7] Thus, the ACF acf_{RBAC} precisely reflects an RBAC policy's access decisions.

In DRBAC models, the semantics of acf_{RBAC} stays the same, however, in contrast to traditional RBAC models, DRBAC model components are not necessarily static anymore and may change with protection state transitions. Thus, additional assignments in dynamic $\sqsupset$ components queried within the ACF, e. g. PA, *roles* or RH, may also change its behavior *directly* leading to additional privileges and, consequently, to altered access decisions (granted instead of denied). Changes in any other dynamic $\sqsupset$ components, e. g. UA (assigning a role to a user) and *user* (performing a login) only make additional privileges *indirectly* possible, thus establish a necessary precondition for that.

Subsequently, we call analysis goals regarding a single $\sqsupset$ model component *simple safety properties*, whereby we differentiate into *level-I* and *level-II* simple safeties, based on the potential to directly or indirectly contribute to a proliferation of privileges. Overall, the following simple safety variants are conceivable, named consistently according to the scheme "$\langle\sqsupset_i$ parameter(s)$\rangle$-$\langle\sqsupset_i$ name$\rangle$":

- $\langle u\rangle$-, $\langle r\rangle$- and $\langle u, r\rangle$-*UA*-safety,
- $\langle p\rangle$-, $\langle r\rangle$- and $\langle p, r\rangle$-*PA*-safety,
- $\langle s\rangle$-, $\langle u\rangle$- and $\langle s, u\rangle$-*user*-safety,
- $\langle s\rangle$-, $\langle r\rangle$- and $\langle s, r\rangle$-*roles*-safety,
- $\langle\cdot, r\rangle$-, $\langle r, \cdot\rangle$- and $\langle r, r'\rangle$-*RH*-safety.[8,9]

[7] Because of the reflexivity property of the partial order RH (see Sec. 3.3), it is correct to assume a role r' that is checked against PA, even in case there is no other role r senior to.

[8] Since sessions are typically created and assigned sometime during an AC system's runtime, note that $\langle s\rangle$-, $\langle s, u\rangle$-, $\langle s\rangle$- and $\langle s, r\rangle$-safety properties only make sense if the model contains a pre-initialized subset of session elements to which these session parameters refer to.

[9] In order to differenciate between safety regarding the superordination or subordination of roles in RH, "$\cdot$" denotes an empty placeholder.

Moreover, simple safeties can be composed, such that multiple dynamic $\beth$ components are taken into account. We refer to these properties as *composed safety properties* and name them straight-forward based on the composed safeties' names, such that for example, $\langle u\rangle$*-user-*$\langle r\rangle$*-roles*-safety is composed of $\langle u\rangle$*-user*-safety and $\langle r\rangle$*-roles*-safety properties.

Complementary to this classification, safety properties can be concretized to application-specific analysis goals by specifying parameter values (e. g. a user, role, permission or session) of $\aleph$ components related to the safety-relevant $\beth$ components.

In the remaining part of this section, we discuss a selection of classified safety properties which are relevant to the modeled HIS security policy from Sec. 3.2 demonstrating the approach to tailor DRBAC safety properties for an analysis scenario of a particular application-specific security policy.

Example 5 (*Tailoring DRBAC Safety Analysis Goals*). Given the examplary HIS DRBAC model from Example 2. We consider the following analysis question: Since in the HIS security policy it is assumed that doctors may *temporarily* delegate treatment capabilities (ward-specific doctor roles such as rDoctorICU, rDoctorCard, or rDoctorMat), is it ever possible, that a doctor is being assigned to at least two ward-specific doctor roles and capable of using those roles actively in some session, and thereby *permanently* maintains their actually separated privileges?

The analysis goal informally described above actually targets two formal model components defined as dynamic: The *UA* relation, in which the permanent assignment of a ward-specific doctor role establishes a necessary precondition, and the *roles* function, in which the activation directly makes the role's privileges usable in a particular session. Thus, starting point for the formal description of the analysis goal is the level-II simple safety regarding *UA* called $\langle r\rangle$*-UA-safety*, which considers for a given role r whether r is ever assigned to some user u.

Definition 5 ($\langle r\rangle$*-UA-Safety*). A DRBAC model $\langle\Gamma,\Sigma,\Delta\rangle$ is $\langle r\rangle$*-UA*-safe w. r. t. a state γ and a role $r\in R_{[\gamma]^*}$ iff there is no input sequence $\sigma_1\ldots\sigma_n\in\Sigma^*$ that, starting with γ leads to γ' via Δ, such that $\exists u\in U_{[\gamma]^*}\cap U_{[\gamma']^*}$: $\langle u,r\rangle\in UA_{\gamma'}$ whereas $\langle u,r\rangle\notin UA_{\gamma}$.[10]

The dynamics of *roles* allow privileges to be proliferated directly (through the activation of roles in a session). These dynamics motivate a level-I safety property regarding *roles* and a role r, to investigate whether, based on a particular model state, a role such as rDoctorICU can be activated in some session. We define this property as the $\langle r\rangle$*-roles-safety* as follows.

Definition 6 ($\langle r\rangle$*-roles-Safety*). A DRBAC model $\langle\Gamma,\Sigma,\Delta\rangle$ is $\langle r\rangle$*-roles*-safe w. r. t. a state γ and a role $r\in R_{[\gamma]^*}$ iff there is no input sequence $\sigma_1\ldots\sigma_n\in\Sigma^*$ that, starting with γ leads to γ' via Δ, such that $\exists s'\in S_{[\gamma']^*}$: $r\in roles_{\gamma'}(s')$ whereas $\nexists s\in S_{[\gamma]^*}$: $r\in roles_{\gamma}(s)$.

The definitions so far only considered a single component isolated and, thus, are rather limited in terms of expressiveness. Since we target a single safety definition which takes both model components, *UA* and *roles*, into account, we define the composed safety property called $\langle r\rangle$*-UA-*$\langle r\rangle$*-roles-safety* as follows.

Definition 7 ($\langle r\rangle$*-UA-*$\langle r\rangle$*-roles-Safety*). A DRBAC model $\langle\Gamma,\Sigma,\Delta\rangle$ is $\langle r\rangle$*-UA-*$\langle r\rangle$*-roles*-safe w. r. t. a state γ and a role $r\in R_{[\gamma]^*}$ iff there is no input sequence $\sigma_1\ldots\sigma_n\in\Sigma^*$ that, starting at γ, leads to γ' via Δ, such that $\exists u\in U_{[\gamma]^*}\cap U_{[\gamma']^*},\exists s'\in S_{[\gamma']^*},u=user_{[\gamma']^*}(s')$: $\langle u,r\rangle\in UA_{\gamma'}\wedge r\in roles_{\gamma'}(s')$ whereas $\nexists s\in S_{[\gamma]^*},u=user_{[\gamma]^*}(s)$: $\langle u,r\rangle\notin UA_{\gamma}\wedge r\in roles_{\gamma}(s)$.

Finally, we can now formally concretize our analysis goal, which was informally established at the beginning, as $\langle r\rangle$*-UA-*$\langle r\rangle$*-roles*-safety with $r\in\{\text{rDoctorICU},\text{rDoctorCard},\text{rDoctorMat}\}$.

4.2 DRBAC Safety Analysis

In the previous section we discussed, given a DRBAC model and an informal safety analysis question, how model-specific safety properties can be derived and formally specified. The purpose of this section is to lay the algorithmic foundations for an automated analysis of DRBAC safety properties and then, using the HIS DRBAC model as an example, to show how analysis results can be applied by policy engineers to correct model errors.

Heuristic-driven Safety Analysis. As has been proven for the seminal HRU model (Harrison et al., 1976), decidable safety properties require the restriction of a model's computational power. Since such restrictions reflect the semantical needs of application-specific policies which usually differ from each other, the corresponding analysis algorithms are also specific to these models. Consequently, new models require analysis algorithms to be built from scratch.

In contrast, our previous work aims at a common generalization of safety analysis algorithms for dynamic AC models based on deterministic automatons (Kühnhauser and Pölck, 2011; Amthor et al., 2013; Amthor, 2016; Amthor, 2017). Instead of restricting a

[10]Using the regular expression $[\gamma]^*$, $R_{[\gamma]^*}$ and $U_{[\gamma]^*}$ describe either static sets R and U or, if they are part of the state space, dynamically changing sets R_γ and U_γ.

```
In: $\Delta$ ... state transition scheme
    $\gamma_0$ ... state to be analyzed
    $target$ ... unsafety target
Out: $stateSeq$ ... states sequence leaking $target$
1  $\gamma \leftarrow \gamma_0$; $stateSeq \leftarrow \gamma$;
2  $CDG \leftarrow$ CDGAssembly($\Delta, \gamma, target$);
3  repeat
4      $inputSeq \leftarrow$ generateInput($CDG$, $\gamma$);
5      while $\sigma \leftarrow inputSeq.next$ do
6          $\gamma' \leftarrow$ transition($\gamma, \sigma, \Delta$);
7          $stateSeq \leftarrow stateSeq \circ \gamma'$;
8          $\gamma \leftarrow \gamma'$;
9  until isUnsafe($\gamma_0, \gamma', target$);
10 return $seq$;
```

Algorithm 1: DEPSEARCH.

model's expressive power to achieve safety decidablity, our approach accepts the impossibility of confirming model to be *safe*. We choose to trade accuracy for tractability: by performing a heuristic-driven simulation of a dynamic AC model, *unsafe* model states may be found (given such exist), while the termination of the algorithm cannot be guaranteed.

Heuristic safety analysis algorithms exploit the semi-decidablity of the problem: Given two model protection states with one state being some follow-up state of the other, it is efficiently decidable whether in the follow-up state a proliferation can be observed. Thus, starting with some state γ, heuristic search algorithms guide a model through its state space by generating input sequences, feeding them into the automaton and checking for each reached state γ' whether it renders state γ unsafe. A successful heuristic must therefore maximize the propability of an input to contribute to a path from γ to γ_{target}.

Our most promising heuristic for the safety analysis of dynamic automaton-based AC models is DEPSEARCH (*dependency search*) (Amthor et al., 2013). Based on the insight that in the most difficult case, privilege proliferation in a dynamic model occur only after specific state transition sequences where each command executed depends exactly on the execution of its predecessor, DEPSEARCH explicitly searches for such dependencies and executes corresponding input sequences. Given the termination of the analysis algorithm and that unsafe states are found, the analysis results provide valuable hints on model correctness and policy engineers are pointed to input sequences that lead to such states.

In the following, we adapt the approach of heuristic safety analysis for DRBAC models and discuss a DRBAC-tailored descendant of DEPSEARCH.

The DEPSEARCH Algorithm for DRBAC. We developed the DEPSEARCH heuristic based on the observation of sequences of interdependent commands leading to proliferations of privileges. Hence, we derived the idea of a two-phases algorithm: First, a static analysis of inter-command dependencies is performed; a subsequent, dynamic analysis consists of state space exploration by simulating the automaton's behavior. We will discuss these phases on an informal basis, for an in-depth discussion and evaluation of DEPSEARCH for the HRU model see (Amthor et al., 2013; Amthor et al., 2014). The generic base algorithm is presented in Alg. 1.

During the first phase (cf. Alg. 1, line 2), a *static analysis* of the model's state transition scheme Δ is performed. It yields a graph-based description of inter-command dependencies, constituted by adding (as a part of POST) and requiring (as a part of PRE) the same element (e. g. $\langle u_{doctor}, \text{rDoctorCard} \rangle$ in UA) in two different commands. As a result of the static analysis, these dependencies are modeled by a *command dependency graph* (CDG) $\langle V, E \rangle$ where nodes $v \in V$ represent commands, and an edge $\langle c_1, c_2 \rangle$ denotes that a post-condition of c_1 matches at least one pre-condition of c_2.

The CDG can be assembled in a way that all paths from vertices without incoming edges to vertices without outgoing edges indicate input sequences for reaching γ_{target} from γ. To achieve this, two virtual commands c_γ and c_{target} are generated: c_γ is the source of all paths in the CDG, since it represents the state γ to analyze in terms of a command specification added to Δ. It is generated by encoding the dynamic $\sqsupset$ components in POST(c_γ). In a similar manner, c_{target} is the sink of all paths in the CDG, which represents all possible states γ_{target} by checking the presence of the target elements in the safety-relevant $\sqsupset$ component according to the safety definition in PRE(c_{target}).

In the second, *dynamic analysis* phase (cf. Alg. 1, lines 3–9), the CDG is used to guide dynamic state transitions by generating input sequences to the automaton. The commands involved in each sequence are chosen according to different paths from c_γ and c_{target}, which in turn determine the direction of state space exploration. Paths in the CDG are generated based on a simple ant algorithm: with each edge traversed to generate an input sequence, the algorithm increases an edge weight "scent" which has repellent effect for the next iteration of the CDG traversal, thus effectively leading to a full path coverage and potentially maximal chances for actually generating a proliferation.

DEPSEARCH successively generates input sequences by traversing the CDG on every possible path and in turn parameterizing the emerging sequence of

```
In: $\gamma = \langle U_\gamma, S_\gamma, UA_\gamma, user_\gamma, roles_\gamma \rangle \ldots$ state to
    be analyzed
    $\gamma' = \langle U_{\gamma'}, S_{\gamma'}, UA_{\gamma'}, user_{\gamma'}, roles_{\gamma'} \rangle \ldots$
    state to check for unsafety
    $target = \langle r_{target} \rangle \ldots$ target role
Out: true iff $\gamma$ is $\langle r \rangle$-$UA$-$\langle r \rangle$-$roles$-unsafe
     w. r. t. $r$, false else
1  $sExists \leftarrow$ false;
2  for $u \in U_\gamma \cap U_{\gamma'}$ do
3      for $s' \in \{st' \in S_{\gamma'} \mid u = user_{\gamma'}(st')\}$ do
4          if $\langle u, r_{target} \rangle \in UA_{\gamma'} \land$
             $r_{target} \in roles_{\gamma'}(s')$ then
5              for $s \in \{st \in S_\gamma \mid u = user_\gamma(st)\}$ do
6                  if $r_{target} \in roles_\gamma(s)$ then
7                      $sExists \leftarrow$ true;
8              if $\langle u, r_{target} \rangle \notin UA_\gamma \land$
                 $sExists =$ false then
9                  return true;
10 return false;
```

Algorithm 2: isUnsafe.

commands with values from Σ_Y. Each effected state transition is simulated by the algorithm, and once a CDG path is completed, the validity of the unsafety-criteria is checked.

The corresponding algorithm used is directly tailored to the unsafety-criterion for the particular analysis. As an example, Alg. 2 represents the $\langle r \rangle$-UA-$\langle r \rangle$-$roles$-unsafety which is defined as follows.

Definition 8 ($\langle r \rangle$-UA-$\langle r \rangle$-*roles*-Unsafety). A DRBAC model $\langle \Gamma, \Sigma, \Delta \rangle$ is $\langle r \rangle$-UA-$\langle r \rangle$-$roles$-unsafe w. r. t. a state γ and a role $r \in R_{[\gamma]^*}$ iff there is an input sequence $\sigma_1 \ldots \sigma_n \in \Sigma^*$ that, starting at γ, leads to γ' via Δ, such that $\exists u \in U_{[\gamma]^*} \cap U_{[\gamma']^*}, \exists s' \in S_{[\gamma']^*}, u = user_{[\gamma']^*}(s')$: $\langle u, r \rangle \in UA_{\gamma'} \land r \in roles_{\gamma'}(s')$ whereas $\nexists s \in S_{[\gamma]^*}, u = user_{[\gamma]^*}(s)$: $\langle u, r \rangle \notin UA_\gamma \land r \in roles_\gamma(s)$.

Interpretation of Analysis Results. We now apply the DEPSEARCH heuristic for the analysis of the HIS DRBAC model (see Example 2) with regards to $\langle r \rangle$-UA-$\langle r \rangle$-$roles$-safety checking for the corresponding unsafety-criterion. In order to keep the algorithm comprehensible and to avoid the problem of dynamic dependencies of command parameters (Amthor and Rabe, 2020), wherever a parameter is passed as a parameter is passed to a dynamic, safety-relevant model component (e. g. r_{ward} in the command assignDoctor), the respective command definition is treated like a macro. This requires that the command be expanded in terms of acceptable values. For example, the command assignDoctor expands for $r_{ward} \in \{\text{rDoctorICU}, \text{rDoctorCard}, \text{rDoctorMat}\}$ to assignDoctorICU, assignDoctorCard, assignDoctorMat).

Then, the execution of DEPSEARCH results in the following sequence of inputs and state transitions, which leads to a violation of the examined safety.

1. $(\gamma_0 \to \gamma_1)$ loginManager(drKelso)
2. $(\gamma_1 \to \gamma_2)$ assignDoctorICU($s \in S_\gamma$: drKelso $= user_\gamma(s)$, drJD)
3. $(\gamma_2 \to \gamma_3)$ assignDoctorCard($s \in S_\gamma$: drKelso $= user_\gamma(s)$, drJD)
4. $(\gamma_3 \to \gamma_4)$ loginDoctorICU(drJD)
5. $(\gamma_4 \to \gamma_5)$ loginDoctorCard(drCox)
6. $(\gamma_5 \to \gamma_6)$ delegateTreatmentDoctorCard($s \in S_\gamma$: drCox $= user_\gamma(s)$, $s \in S_\gamma$: drJD $= user_\gamma(s)$, rDoctorCard)

According to the MSPE process, the error be subject to correction of the model by a qualified policy engineer. To ensure that, at any given point in time, only a single ward-specific doctor role is *permanently* associated with each doctor user, the PRE part of the assignDoctor command would have to check whether a user associated with a ward-specific doctor role already has such a role. This can be achieved by adding a condition of the following kind:

$$\forall r \in \{\text{rDoctorICU}, \text{rDoctorCard}, \text{rDoctorMat}\} : \langle u_{\text{doctor}}, r \notin UA_\gamma \rangle .$$

Alternatively, in the POST part of the assignDoctor command, the already associated ward-specific doctor role could be de-associated, and, if activated in any of that user's sessions, the "former" ward doctor role must be deactivated and the "new" ward doctor role be activated. Since this kind of proliferation can also occur in the command assignNurse, an analogous correction is necessary.

5 CONCLUSIONS

This paper presents DRBAC, a modeling scheme for RBAC policies supporting their safety analysis by protection state dynamics tailorable to a plethora of application-specific safety analysis goals. We demonstrate the potential of our approach based on an exemplary healthcare information system security policy.

By supporting both policy specification and analysis, we consider our approach as a step towards streamlining the process of model-based security policy engineering. Future work will apply this approach to the more generalized class of ABAC policies. To extend our approach even further to the step of model implementation, ongoing work focuses on the compiler-automated translation of model specifications into source code, and secure runtime environments for rigorously enforcing embedded policies within operating system and application security architecture implementations.

REFERENCES

Amthor, P. (2016). The Entity Labeling Pattern for Modeling Operating Systems Access Control. In *E-Business and Telecomm.: 12th Int. Joint Conf., ICETE 2015, Revised Selected Papers*, pages 270–292.

Amthor, P. (2017). Efficient Heuristic Safety Analysis of Core-based Security Policies. In *Proc. 14th Int. Conf. on Secur. and Cryptogr.*, pages 384–392.

Amthor, P., Kühnhauser, W. E., and Pölck, A. (2013). Heuristic Safety Analysis of Access Control Models. In *Proc. 18th ACM Symp. on Access Control Models and Technol.*, pages 137–148.

Amthor, P., Kühnhauser, W. E., and Pölck, A. (2014). WorSE: A Workbench for Model-based Security Engineering. *Comp. & Secur.*, 42(0):40–55.

Amthor, P. and Rabe, M. (2020). Command Dependencies in Heuristic Safety Analysis of Access Control Models. In *Found. and Practice of Secur.*, vol. 12056 of *LNCS*, pages 207–224.

Barker, S. (2009). The Next 700 Access Control Models or a Unifying Meta-Model? In *Proc. 14th ACM Symp. on Access Control Models and Technol.*, pages 187–196.

Basin, D., Clavel, M., and Egea, M. (2011). A Decade of Model-Driven Security. In *Proc. 16th ACM Symp. on Access Control Models and Technol.*, pages 1–10.

Calzavara, S., Rabitti, A., and Bugliesi, M. (2015). Compositional Typed Analysis of ARBAC Policies. In *Proc. IEEE 28th Comp. Secur. Found. Symp.*, pages 33–45.

Chakraborty, S., Sandhu, R., and Krishnan, R. (2020). On the Feasibility of RBAC to ABAC Policy Mining: A Formal Analysis. In *Proc. 8th Int. Conf. on Sec. Knowl. Managem. in Artific. Intell. Era*, pages 147–163.

Dinh, K. K. Q., Tran, T. D., and Truong, A. (2017). Security Analysis of Administrative Role-Based Access Control Policies with Contextual Information. In *Proc. 4th Int. Conf. on Future Data and Secur. Eng.*, vol. 10646 of *LNCS*, pages 243–261.

Fernández, M., Mackie, I., and Thuraisingham, B. (2019). Specification and Analysis of ABAC Policies via the Category-Based Metamodel. In *Proc. 9th ACM Conf. on Data and App. Secur. and Priv.*, pages 173–184.

Ferraiolo, D., Kuhn, D. R., and Chandramouli, R. (2007). *Role-Based Access Control*. Artech House. Sec. Ed., ISBN 978-1-59693-113-8.

Harrison, M. A., Ruzzo, W. L., and Ullman, J. D. (1976). Protection in Operating Systems. *Comm. of the ACM*, 19(8):461–471.

Jayaraman, K., Tripunitara, M., Ganesh, V., Rinard, M., and Chapin, S. (2013). MOHAWK: Abstraction-Refinement and Bound-Estimation for Verifying Access Control Policies. *ACM Trans. on Inform. and Syst. Secur.*, 15(4):18:1–18:28.

Jha, S., Li, N., Tripunitara, M., Wang, Q., and Winsborough, W. (2008). Towards Formal Verification of Role-Based Access Control Policies. *IEEE Trans. on Depend. Secure Comp.*, 5:242–255.

Jin, X., Krishnan, R., and Sandhu, R. (2012a). A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In *Proc. 26th Ann. IFIP WG 11.3 Conf. on Data and App. Secur. and Priv.*, vol. 7371 of *LNCS*, pages 41–55.

Jin, X., Sandhu, R., and Krishnan, R. (2012b). RABAC: Role-Centric Attribute-Based Access Control In *Proc. 6th Int. Conf. on Math. Methods, Models and Arch. for Comp. Netw. Secur.*, vol. 7531 of *LNCS*, pages 84–96.

Kühnhauser, W. E. and Pölck, A. (2011). Towards Access Control Model Engineering. In *Proc. 7th Int. Conf. on Inform. Syst. Secur.*, pages 379–382.

Li, N. and Tripunitara, M. V. (2006). Security Analysis in Role-Based Access Control. *ACM Trans. on Inform. and Syst. Secur.*, 9(4):391–420.

Pölck, A. (2014). *Small TCBs of Policy-controlled Operating Systems*. Universitätsverlag Ilmenau.

Ranise, S., Truong, A., and Armando, A. (2014). Scalable and Precise Automated Analysis of Administrative Temporal Role-Based Access Control. In *Proc. 19th ACM Symp. on Access Control Models and Technol.*, pages 103–114.

Sandhu, R., Bhamidipati, V., and Munawer, Q. (1999). The ARBAC97 Model for Role-based Administration of Roles. *ACM Trans. on Inf. Syst. Secur.*, 2(1):105–135.

Sandhu, R., Ferraiolo, D., and Kuhn, R. (2000). The NIST Model for Role-Based Access Control: Towards a Unified Standard. In *Proc. 5th ACM Workshop on Role-Based Access Control*, pages 47–63.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-Based Access Control Models. *IEEE Comp.*, 29(2):38–47.

Shahen, J., Niu, J., and Tripunitara, M. (2015). Mohawk+T: Efficient Analysis of Administrative Temporal Role-Based Access Control (ATRBAC) Policies. In *Proc. 20th ACM Symp. on Access Control Models and Technol.*, pages 15–26.

Stoller, S. D., Yang, P., Gofman, M., and Ramakrishnan, C. R. (2011). Symbolic Reachability Analysis for Parameterized Administrative Role Based Access Control. *Comp. & Secur.*, 30(2-3):148–164.

Stoller, S. D., Yang, P., Ramakrishnan, C. R., and Gofman, M. I. (2007). Efficient Policy Analysis for Administrative Role Based Access Control. In *Proc. 14th ACM Conf. Comp. & Comm. Secur.*, pages 445–455.

Tripunitara, M. V. and Li, N. (2007). A Theory for Comparing the Expressive Power of Access Control Models. *Jour. of Comp. Secur.*, 15(2):231–272.

Vimercati, S. D. C. d., Samarati, P., and Jajodia, S. (2005). Policies, Models, and Languages for Access Control. In *Proc. 4th Int. Conf. on Databases in Networked Inform. Syst.*, vol. 3433/2005 of *LNCS*, pages 225–237.