

The Missing Piece of the ABAC Puzzle: A Modeling Scheme for Dynamic Analysis

Marius Schlegel^a and Peter Amthor^b

Technische Universität Ilmenau, Germany

Keywords: Security Engineering, Security Policies, Attribute-based Access Control Models, ABAC, Security Models, Reachability Property, Dynamic Analysis, Heuristic Safety Analysis, Formal Methods.

Abstract: Attribute-based access control (ABAC) has made its way into the mainstream of engineering secure IT systems. At the same time, ABAC models are still lagging behind well-understood, yet more basic access control models in terms of dynamic analyzability. This has led to a plethora of methods, languages, and tools for designing and integrating ABAC policies, but only few to formally reason about them in the process. We present DABAC, a modeling scheme to pick up that missing piece and put it right into its place in the security engineering workflow. Based on an automaton calculus, we demonstrate how DABAC can be leveraged as a holistic formal basis for engineering ABAC models, analyzing their dynamic properties, and providing a functional specification for their implementation. This sets the stage for comprehensive tool support in building future ABAC systems.

1 INTRODUCTION


Attribute-based access control (ABAC) has established as the de-facto paradigm for AC models (Jin et al., 2012; Hu et al., 2014; Ferraiolo et al., 2016; Fernández et al., 2019). ABAC solves a number of issues of popular models regarding expressiveness, granularity, scalability (cf. role explosion in large-scale RBAC models), and context-awareness. Authorizations depend on user, object, and environment attributes involved in access requests; decisions may change if attribute values change. This makes ABAC suitable for a broad range of different application scenarios, such as grid computing (Lang et al., 2009), IoT (Bhatt and Sandhu, 2020), as well as large and distributed information systems (Mukherjee et al., 2017).


However, this flexibility also imposes challenges: Modeling, specification, analysis, and implementation of ABAC policies are not trivial. In order to lower the hurdles for using ABAC, the AC community contributed many important pieces to this comprehensive and complex puzzle: Specialized policy engineering and mining approaches (Narouei et al., 2017; Xu and Stoller, 2015), the NIST standardization (Hu et al., 2014), modeling schemes (also called meta-models in the literature) (Jin et al., 2012; Fernández et al.,

2019; Bertolissi et al., 2020) as well as numerous application-specific ABAC models form a comprehensive framework for modeling ABAC policies. In addition, XACML and NGAC provide support for the specification of ABAC policies and security architectures for their enforcement (Ferraiolo et al., 2016).

Since security policies implement an AC system's security requirements, their correctness is highly critical. A modern approach for policy engineering founded on formal methods is model-based security policy engineering (MSPE) (Vimercati et al., 2005; Tripunitara and Li, 2007; Basin et al., 2011). As a specialization of generic software engineering, MSPE is divided into three steps: (1.) in model engineering, a *model* as a formal policy representation and its correctness conditions (*model properties*) are defined; (2.) during model analysis, formal methods are applied to verify the former against the latter; (3.) finally, in model implementation, the model verified so far is translated into actual code for implementation.

Decades of experience and best practices have yielded *modeling schemes*: formal toolboxes of predicates, set-theoretical expressions, automata theory etc., that support engineers in choosing model formalisms suitable for policy specification and implementation as well as the analysis of correctness properties, e. g. for ABAC (Ahmed and Sandhu, 2017; Jha et al., 2019; Marin et al., 2019). Probably one of the most fundamental and challenging properties are dynamic state

^a  <https://orcid.org/0000-0001-6596-2823>

^b  <https://orcid.org/0000-0001-7711-4450>

reachability properties which can be used to foresee privilege escalation vulnerabilities of a policy (also known as safety). Generally, those properties require dynamic analyses which are pestered by computational complexity (Tripunitara and Li, 2007). Moreover, such properties need to be (re-)defined for any application-specific model. Since this 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 (Bertolissi et al., 2020).

We aim at contributing to these efforts with a focus on two sub-problems of the former: experience teaches us that (1.) dynamic changes in an AC system may arise from both administrative operations and user operations; (2.) generally, interesting dynamic properties that should be subject to model analysis queries tend to be undecidable (cf. safety). The main contribution of this paper is DABAC – a modeling scheme, that (1.) models protection state dynamics in an automaton-based formalism, which enables the analysis of undecidable model properties and unlocks ABAC models for a broad range of existing model analysis approaches which have been developed for the class of automaton-based AC models; (2.) while preserving the expressiveness of state-of-the-art ABAC models already pursuing a future formal standard. The paper is organized as follows: After giving an overview of related work in § 2, we present our DABAC modeling scheme and ready-made patterns for defining privilege escalation (safety) properties in § 3. In § 4, we discuss the application of DABAC to an IoT healthcare scenario following an MSPE workflow consisting of model engineering and analysis.

2 RELATED WORK

This section reviews related work in modeling schemes for ABAC policies. We will summarize them in context of the goals of DABAC. For a more technical comparison with $ABAC_{\alpha}$ and C-ABAC, refer to § 3.2.4.

ABAC $_{\alpha}$ (Jin et al., 2012). The original motivation of $ABAC_{\alpha}$ is to demonstrate how a semantically minimal ABAC modeling scheme can be used to subsume the traditional AC model paradigms of identity-based AC, multi-level security, and RBAC. However, $ABAC_{\alpha}$ was also found useful as an ABAC formalism for practical model engineering and safety analyses (cf. (Ahmed and Sandhu, 2017)). Due to the deliberate restriction of model semantics to the necessary minimum, $ABAC_{\alpha}$ provides many predefined abstractions

for typical policies from the above-mentioned families, however without aiming at the full range of highly diverse ABAC applications and their AC policies.

CBAC (Barker, 2009). Being a widely recognized calculus to generalize arbitrary AC models, CBAC takes a different approach than $ABAC_{\alpha}$. The modeling scheme is based on a highly abstract, axiomatic concept of categorized entities and their relationships, that can be used to represent attributes and attribution mechanisms of an ABAC policy. Because of its high degree of abstraction, CBAC imposes a significant semantical gap to both model engineering and model implementation. With a strong focus on the MSPE workflow, we choose to represent ABAC policies closer to their functional semantics instead; however, parts of our terminology are based on the commonly adopted terms of CBAC abstractions.

C-ABAC (Fernández et al., 2019). C-ABAC is an ABAC modeling scheme that results from an application of CBAC to ABAC policies. C-ABAC and DABAC both adopt the idea of *environmental information* as often required by ABAC applications. Even beyond, both modeling schemes aim at generality, which results in a comparable expressive power with one exception: other than DABAC, C-ABAC explicitly excludes dynamic protection state changes from the modeled semantics. Using the same formal abstractions as CBAC, the latter serves as a solid foundation to verify static model properties rather than to enable dynamic analysis and functional policy specification.

Admin-CBAC (Bertolissi et al., 2020). Admin-CBAC reuses C-ABAC abstractions to model dynamics, based on the use case of AC system administration. This enables analyses of reachability properties that allows to reason about safety (amongst others). However, as the authors note, Admin-CBAC deliberately restricts model analysis to decidable instances of these properties (Bertolissi et al., 2020, Abstract). With the aim to serve as a basis for both simulative and heuristic methods that address exactly this gap in model analysis, DABAC departs the axiomatic approach from CBAC, C-ABAC, and Admin-CBAC in favor of an automaton-based modeling scheme. Moreover, instead of separating administrative from non-administrative models through different modeling schemes, we opted for integrating any administrative dynamics with such from regular user operations. As mentioned, this potentially results in more complex (up to undecidable) dynamic properties to analyze, but allows for a more holistic view on the modeled policy on the other hand.

We have motivated and discussed this paradigm in previous work on RBAC (Schlegel and Amthor, 2020).

3 MODELING SCHEME

This section presents DABAC, a formal ABAC modeling scheme suitable for simulative model analysis, and, based on their results, iterative model refinement as part of the MSPE workflow. We start by analyzing the requirements (§ 3.1), then address the individual abstraction levels defined by the modeling scheme (§ 3.2): its primitive models components (§ 3.2.1), its definition of model dynamics (§ 3.2.2), and its design space for defining model safety (§ 3.2.3). Finally, we discuss the expressive power of DABAC (§ 3.2.4).

3.1 Requirements

ABAC Expressiveness. From an ABAC application’s perspective, we require any modeling scheme to be both sufficiently expressive and usable in a streamlined manner to express typical ABAC policies from real-world scenarios, such as in the application domains of vehicle networks (Gupta et al., 2020), cloud and edge computing (Bhatt and Sandhu, 2020), or hospital information systems (Ray et al., 2017; Mukherjee et al., 2017). More specifically, this means that a DABAC model should be capable of expressing an ABAC feature typical for the above-mentioned, usually distributed, application scenarios: the need to evaluate both an internal state of the AC system as well as external information for making an access decision. Typical examples for this, also called *environmental information* in the literature (Barker, 2009; Fernández et al., 2019), are notifications from third-party systems, sensor values that depend on physical conditions, or threat information for risk management. These can be observed from external sources, and thus need to be modeled as distinct entities, attributes and attribute values, respectively. While such information might be used in access decisions, an AC system has no influence on their possible changes.

Model Engineering. Flexibility of modeling abstractions is a key requirement to support the MSPE core concept of reiterative model refinement. For modeling ABAC policies, this primarily relates to abstractions for attribution and attribute evaluation. In this regard, DABAC should support (1.) indirect attribution (attributes which are themselves subject to attribution, i. e. role attributes in *parameterized RBAC* models (Stoller et al., 2009)), and (2.) both enumeration-based and formula-based attribute evaluation (Biswas et al.,

2016). It should be noted that this flexibility is already required by the MSPE core concept of reiterative model refinement, based on both refined policy requirements (as part of model engineering) and revised policy rules (as an outcome of model analysis).

Model Analysis. To reason about dynamic properties of ABAC policies, we settle for a family of heuristic algorithms that simulate dynamic model behavior, which allow to study undecidable *safety*-properties (Amthor et al., 2014). We therefore require an automaton calculus, which has been successfully used for this purpose in previous work (Amthor et al., 2013; Amthor, 2017; Schlegel and Kühnhauser, 2020). Based on this calculus, DABAC should be able to support custom, policy-specific safety definitions by using the model’s abstractions. Similar to how ABAC generalizes RBAC, this enables reusing previous work on safety analysis methods for dynamic RBAC models (Schlegel and Amthor, 2020).

Model Implementation. As a last requirement, second to their benefits in model analysis, DABAC models should be re-usable as a formal specification for implementing the actual AC system. This means that, based on model abstractions, a functional specification of interface, internal state and possibly external state of the AC system should be possible. This lowers semantical hurdles towards planned semi-automatic implementation of policy decision making and even runtime monitoring of policy invariants, both parts of future work (cf. § 5).

3.2 DABAC

As a modeling scheme rather than a specific model in itself, DABAC defines how a specific ABAC policy should be formalized. This happens on three semantical levels of abstraction:

1. *Primitive Model Components:* provide formal artifacts to express a policy’s complete authorization semantics, which are based on families for
 - sets of elementary identifiers to express entities and attribute values;
 - their interrelations, formalized as relations, mappings, or boolean expressions, to express attribution as well as authorization rules.
2. *Model Dynamics:* rely on the idea of a specialized automaton model $\langle \Gamma, \Sigma, \Delta, Perm \rangle$ consisting of
 - the *state space* Γ : a set of protection states;
 - the *input set* Σ : a set of possible inputs that may trigger state transitions, based on parameterized operations made available by the AC system;

- the *state transition scheme (STS)* $\Delta \subseteq \Sigma \times \mathcal{P}(\text{Perm}) \times \Phi$: state transition pre- and post-conditions for any input $\sigma \in \Sigma$;¹
 - the *permissions set Perm*: set of identifiers for fine-grained access conditions.
3. The *safety property* as a formal definition of our analysis question the model should be checked against according to our requirements.

It should be noted that during practical model engineering, these levels of abstractions might be addressed in sequence, but due to their semantical interrelations (e. g. a safety property depending on both primitive model components and state definition), they are inherently subject to iterative model engineering. In favor of a concise discussion however, we will now address these three levels in sequence.

3.2.1 Primitive Model Components

To make an authorization decision, any ABAC policy most generally demands three classes of information: which entities are involved in an access, which attributes are associated with these entities, and which access rules should be evaluated to authorize any operation. DABAC covers these information in four families for each primitive model component. In the following, we will informally introduce these families, which are formally defined in Tab. 1.

Entities (EN): a set of identifiers for atomic abstractions immediately involved in an access, either actively or passively (such as users, principals, applications, sessions, devices);

Attribute Values (AV): a set of identifiers that are evaluated in access rules (such as age of a user, login time of a session, numerical sensor value);

Attribute Association (AA): a mapping from either an EN or an AV to either an AV (attribution with a single value) or its power set (attribution with multiple values);

Access Rule (AR): a set containing a logical formula or an attribute enumeration for each permission in *Perm*, which decides for a vector of AV elements as arguments if entities attributed with these values are authorized for that respective permission.

We assume any sets of identifiers to be countable but possibly infinite. For unambiguity, we use the following conventions for notation: In a formal context, we denote the sets representing these families by their italic symbol. When part of prose, we use upright abbreviations as follows: “an EN” stands for “a component from the entities family”, “element of an EN”

stands for “an element in a set which is an EN”, and so on for the other families.

Formally, we treat these families as indexed sets of primitive model components, each a formal artifact required by a concrete policy. The choice of DABAC families explicitly aims at simple, easy to understand ABAC policy semantics in order to support a reiterative model engineering process, which equally involves experts in MSPE as well as in the application domain.

Note that our definition of AV and AA deliberately allows for not only direct attribution, but also to use AV elements as arguments for AA, which satisfies the requirement of iterative model refinement through indirect attribution. Based on (Amthor, 2019), we introduce an *indirection degree* i for each attribute association AV_i , which denotes if the respective domain of these mappings is from EN (direct attribution) iff $i = 0$, or from AV_i (indirect attribution) iff $0 < i \leq i_{\max}$ (where i_{\max} denotes the maximum indirection degree in a model). Whenever used without an index, the respective set symbol denotes any possible indirection degree, i. e. $AV = \bigcup_{0 \leq i \leq i_{\max}} AV_i$ and similar for AA.

While the above description of AR already allows for arbitrary representations of attribute evaluation – both enumeration-based and formula-based – the application-specific requirement of environmental information leads to another, complementary distinction of primitive model components.

Another requirement not addressed so far relates to an explicit distinction between internal and external domain. For a modeled AC system, the internal domain (also called authorization domain) contains any local information and any authorization rules of the AC system. The external domain (information domain) contains observable, external information, which cannot be controlled by the AC system: it cannot effect, anticipate, or causally explain any changes in the external domain. Formally, we demand that any primitive model component is exclusively member of either the internal or the external domain, which results in a classification orthogonal to the semantical families introduced so far. We thus refine them as follows:

EN: are internal entities, whose creation, destruction, and modification solely occurs within the domain of the AC system;

EN^{ext}: are external entities, which are observable at the AC system’s interface to another domain (controlling their observability);

AV: are internal attribute values assigned through mechanisms implemented in the AC system;

AV^{ext}: are external attribute values, whose assignments are observable by the AC system;

¹ Φ denotes the set of expressions in first-order logic.

Table 1: Summary of DABAC families. Indices $n, m \in \mathbb{N}$ are independent for each row.

	internal	external
EN	$EN = \{E_1, \dots, E_n\}$, where $E_{1, \dots, n}$ are policy-specific sets of internal entities	$EN^{\text{ext}} = \{E_{n+1}, \dots, E_{n+m}\}$, where $E_{n+1, \dots, n+m}$ are policy-specific sets of external entities
AV	$AV_i = \{V_1, \dots, V_n\}$, where $V_{1, \dots, n}$ are policy-specific sets of internal attribute values of indirection degree $0 \leq i \leq i_{\max}$	$AV_i^{\text{ext}} = \{V_{n+1}, \dots, V_{n+m}\}$, where $V_{n+1, \dots, n+m}$ are policy-specific sets of internal attribute values of indirection degree $0 \leq i \leq i_{\max}$
AA	$AA_0 = \{att : E \rightarrow V \mid E \in EN \wedge (\exists V' \in AV_0 : V = V' \vee V = \mathcal{P}(V'))\}$, where att are internal direct attribute mappings; $AA_i = \{att : V_{i-1} \rightarrow V_i \mid V_{i-1} \in AV_{i-1} \wedge (\exists V'_i \in AV_i : V_i = V'_i \vee V_i = \mathcal{P}(V'_i))\}$, where att are internal attribute mappings of indirection degree $0 < i \leq i_{\max}$	AA_i^{ext} are external attribute mappings analogous to AA_i , $0 \leq i \leq i_{\max}$
AR	$AR = \{auth_p : [v_1, \dots, v_{n_p}] \rightarrow \mathbb{B}, n_p \in \mathbb{N} \mid p \in Perm \wedge (\exists V_k \in AV \cup AV^{\text{ext}} : v_k \in V_k \vee v_k \in \mathcal{P}(V_k), 1 \leq k \leq n_p)\}$, where $auth_p$ are authorization functions for each permission p , which map a vector of (internal or external) AV elements to a boolean decision value	<i>not applicable</i>

AA, AA^{ext} : are attribution mappings for internal/external attribute associations analogous to AV/AV^{ext} ;

AR: describe system functionality rather than information. To this end, a distinction based on controllability vs. observability is not applicable. Still, however, since authorization is internal functionality of the modeled system, we will technically treat them as part of the internal domain.

3.2.2 Model Dynamics

Being the key-enabler for safety analysis, the deterministic automaton is formalized in the next step.

State Space. As the automaton's state space describes model components subject to runtime mutability, we refrain from defining it on a modeling scheme level here. Instead, we generalize state-specific instances of primitive model components from the families EN and AA as follows:

$$\Gamma = \prod_{k \in K} \mathcal{P}(E_k) \times \prod_{l \in L} \{att_{l\gamma} : X \rightarrow V \mid att_l : X' \rightarrow V \wedge X \subseteq X'\}$$

where $E_k \in EN$ and $K \subseteq \{1, \dots, |EN|\}$ is an index enumeration of dynamic internal entities sets; $att_l \in AA$ and $L \subseteq \{1, \dots, |AA|\}$ is an index enumeration of dynamic internal attribute mappings. For addressing model components in a state $\gamma \in \Gamma$, we will use γ as a subscript, i. e. $\gamma = \langle E_{1\gamma}, \dots, E_{|K|\gamma}, att_{1\gamma}, \dots, att_{|L|\gamma} \rangle$. Similarly we will use the regular expression $[\gamma^*]$ as a subscript of dynamics-agnostic model components, such that $E_{k[\gamma^*]} = E_{k\gamma}$ in a DABAC model where $k \in K$, but $E_{k[\gamma^*]} = E_k$ otherwise.

The rationale for this is twofold. First, for the mutable portion of an ABAC policy, there is no security-relevant use case for creating or deleting elements of an AV, since entities are the sole endpoint abstraction for any access decision. Attribute values, on the other hand, are relevant for such decision only as far as they are assigned to entities – which implies that AA may be mutable in order to model dynamic AC system reconfiguration. AR, finally, must be immutable to prevent a self-modifying *policy* (in contrast to our goal, a dynamically modifyable model state). Moreover, since $auth_p \in AR$ are defined based on AV elements (for both enumerated and formula-based ABAC), the immutable nature of the latter formally requires the former to be immutable as well.

Second, according to the definition of Γ , model components from the external domain must never be part of the automaton's state space. The reason for this is a direct consequence from their rationale as a model abstraction: Since an AC system cannot control, i. e. create, delete, or modify, such components, we cannot alter (nor even formalize) their mutability rules (cf. Δ) to allow for the correction of policy errors that dynamic analyses might expose. Nevertheless, since external components might still impact the authorization of policy-specific operations (which we will formalize as logical pre-conditions in Δ), they must not be ignored. This results in a deliberate decision not to model any external state, but instead treat external model components similar to static, internal components, just without any guaranteed assumptions w. r. t. to their dynamics.

Consequently, the dynamic portion of the policy might be expressed by simply selecting mutable primitive model components from EN and AA .

State Transitions. To formalize (1.) how authorization decisions are made and (2.) how mutable model components change, we discuss next how to define the automaton’s interface Σ and STS Δ . Σ , representing parameterized input operations, and Δ , representing state-change rules, both rely on the definition of AR, which also yields the set of possible permissions $Perm$.

Formally, we define $Perm$ as a set of atomic identifiers and Σ as follows:

$$\Sigma = \{ \langle op, [a_1, \dots, a_{n_{op}}] \rangle \mid op \in OP \wedge (\exists A_k \in EN \cup EN^{ext} \cup AV \cup AV^{ext} : a_k \in A_k, 1 \leq k \leq n_{op}) \}$$

where OP is a set of atomic identifiers for operations made available by the AC system and $[a_k]$ is a vector of operation-specific argument values from either EN or AV, both internal or external. This way, the automaton’s interface is policy-agnostic w. r. t. the semantics of possible access operations.

To finally address the two modeling tasks stated above, we express both pre-conditions (and thus authorization decisions) and post-conditions (changes of mutable components) through a set of tuples $\Delta \subseteq \Sigma \times \mathcal{P}(Perm) \times \Phi$, where each $\langle \sigma = \langle op, [a_k]_{k=1}^{n_{op}} \rangle, Perm_{op}, \phi_{op} \rangle$ is written as:

$$\begin{aligned} &\blacktriangleright \text{op}(a_1, \dots, a_{n_{op}}) ::= \\ &\text{VAR: } v_1 = \phi_1, \dots, v_{m_{op}} = \phi_{m_{op}} \\ &\text{PRE: } \bigwedge_{p \in Perm_{op}} \text{auth}_p([v_{pk}]_{k=1}^{n_p}) \\ &\text{POST: } \phi_{op} \end{aligned}$$

This notation lists a CNF of AR as the pre-condition (abbreviated $op.PRE$) and a boolean expression ϕ_{op} as the post-condition (abbreviated $op.POST$) of any state transition to be authorized via operation op . On an automaton level, this means that $op.PRE$ restricts which states γ to legally transition from, while $op.POST$ defines any differences between γ and the state γ' reachable by any input word σ . Finally, ϕ_{op} consists of redefinitions of one or more mutable primitive model components, or just of “true” in case of not state-modifying operations (i. e. $\gamma = \gamma'$). Note that, while not required for the same expressive power, the additional section VAR supports model engineers as well as model readers by explicitly naming (individual or sets of) attribute values $v_{1, \dots, m_{op}}$ extracted from γ through definitions $\phi_{1, \dots, m_{op}} \in \Phi$ which are then used in $op.PRE$ (such that indices pk are from $\{1, \dots, m_{op}\}$ if v_{pk} is not a constant) or in $op.POST$. See Fig. 2 for some examples.

One may observe that we do not define an output function. The reason for this is that any safety property always aims at changes in potentially reachable protection states – which only state-modifying operations might produce. However, for actually implementing a

DABAC model in the last MSPE step, one may easily derive its access control function (ACF), even without an explicitly formalized output function, based on AR. For this purpose, we explicitly allow the model engineer to include not state-modifying operations in the STS (in contrast to e. g. classical HRU models for safety analysis (Harrison et al., 1976)).

3.2.3 Model Safety

We conclude the discussion of DABAC formalisms with the actual goal of dynamic analysis. As already introduced, the modeling scheme aims at a generalization of the reachability property for protection states that impact the ACF of the model, which might lead to any form of privilege escalation in a model’s terms.

Similar to how an access matrix containing access rights formalizes an ACF, we defined AR containing permission authorization functions to formalize the same for a DABAC model. To this end we propose a most general DABAC safety definition based on the term of *permission leaks*.

Definition 1 (Permission Leak). For two states γ and γ' of a DABAC model, γ' leaks a permission $p \in Perm$ w. r. t. γ iff there is either (1.) an $X \in EN$ and an $att \in AA_0$ (direct leakage) or (2.) an $X \in AV$ and an $att \in AA_i, i > 0$ (indirect leakage), such that

$$\begin{aligned} \exists \{x_1, \dots, x_{n_p}\} \subseteq X_{[\gamma]^*} \cap X_{[\gamma']^*} : \\ \text{auth}_p([att_{[\gamma]^*}(x_h)]_{h=1}^{n_p}) = \text{false} \\ \wedge \text{auth}_p([att_{[\gamma']^*}(x_h)]_{h=1}^{n_p}) = \text{true}. \end{aligned}$$

Some observations towards this definition should be discussed first. (1.) Potentially, both static and dynamic model components could be subject of a privilege escalation, since even a static entity set might be subject to changing attribution. The only exception is a model in which an attribute mapping in AA_0 is static, which formally requires its domain set E to be static as well. Consequently, we use the dynamics-agnostic notation $[\gamma^*]$ as introduced before. (2.) Likewise, both direct and indirect attribute mappings must be taken into account, since auth_p definitions are not restricted to require only direct attribute values. (3.) Since privilege escalation is only relevant (and fixable) for the internal domain of our modeled system, safety definitions only include internal model components for the violating state γ' . External components could still contribute to a path in the model’s state space for γ' to be reachable, though. (4.) We deliberately ask for leaks effected by such model component values x_h that are already present in γ . The reason for this is practical rather than technical: We agree with (Tripunitara and Li, 2013) that a meaningful safety analysis result

should expose entities or indirect attribute values that a model analyst can identify in γ and therefore relate to some actual vulnerability in terms of her policy.

Def. 1 leads to our model-independent definition of DABAC safety.

Definition 2 ($\langle p \rangle$ -Safety). For a state γ of a DABAC model and a permission $p \in Perm$, is there an input sequence $\sigma_1, \dots, \sigma_n \in \Sigma^*$ that leads from γ to γ' via Δ such that γ' leaks p w. r. t. γ ?

If no such sequence exists, γ is called $\langle p \rangle$ -safe.

In natural prose, we ask “Is it possible that some entity without a specific permission gains this permission in the future?” Note that this subsumes the question “Is it possible that an operation op , which evaluates to $op.PRE = \text{false}$ for some arguments, evaluates to $op.PRE = \text{true}$ for the same arguments in the future?”, which might be asked from a practitioner’s view. For any concrete DABAC model, the final step of model engineering is to adapt Defs. 1 and 2 to model-specific EN and AA, which paves the way for a subsequent, tool-based analysis.

As with classical safety, both definitions might be further refined (cf. (Tripunitara and Li, 2013)). The idea here is to add, additionally to p , model-specific parameters that confine our analysis cases to relevant model component values (analysis *targets*) x_n . These could be security-critical objects, untrusted users and the like; again, we support iterative MSPE by allowing for a step-by-step increase of granularity here – starting from Def. 2, which only relates to a state γ and a permission p (in the notation “ $\langle parameter(s) \rangle$ -safe”).

Wrapping up, a parameterized analysis query amends the model-specific safety definition as follows.

Definition 3 (Safety Analysis Query). A DABAC safety analysis query is a tuple $\langle \gamma_0, p, [\tau_1, \dots, \tau_n] \rangle$, where $\gamma_0 \in \Gamma$ is a model state to analyze, p is the target permission to analyze, and $[\tau_k]_{k=1}^n, \tau_k \in EN \cup AV$ is a vector of additional analysis targets.

Composed Safety. For practical policy analysis, we expect an even more specialized safety question relating to future conditions for authorizing a complete operation instead of granting a particular permission. Again, such questions might be expressed through another refinement step, which results in a *composed safety* definition for a corresponding, specialized analysis query. Composed safety can be regarded as a conjoining macro over multiple, operation-specific instances of $\langle p \rangle$ -safety for a specific policy.

We hence conclude this discussion with a definition of such a macro to query composed safety for a fully parameterized input operation.

Definition 4 ($\langle op \rangle$ -Safety Analysis Query). An $\langle op \rangle$ -safety analysis query is a tuple $\langle \gamma_0, op, [x_1, \dots, x_{n_{op}}] \rangle$, where $\gamma_0 \in \Gamma$ is a model state to analyze, op is the target operation to analyze, and $x_1, \dots, x_{n_{op}}$ are actual arguments of op .

Definition 5 ($\langle op \rangle$ -Safety). Given an $\langle op \rangle$ -safety analysis query $q = \langle \gamma_0, op, [x_1, \dots, x_{n_{op}}] \rangle$ against a DABAC model such that $\langle \langle op, [a_k]_{k=1}^{n_{op}} \rangle, Perm_{op}, \phi_{op} \rangle \in \Delta$. Is there an input sequence $\sigma_1, \dots, \sigma_n \in \Sigma^*$ that leads from γ_0 to γ' via Δ such that both of the following is true:

- $\phi_{op} = \text{true}$ for the input $\langle op, [x_k]_{k=1}^{n_{op}} \rangle$ in state γ' and
- $\exists p \in Perm_{op} : \langle p \rangle$ -safety for the query $\langle \gamma_0, p, [x_k]_{k=1}^{n_{op}} \rangle$ is false?

If no such sequence exists, $\langle op \rangle$ -safety for q is true.

After having defined model components and model dynamics for an application scenario, we end up with a model-specific safety property and analysis query. These pieces will be put together be an exemplary MSPE workflow illustrated in § 4.

3.2.4 Expressive Power of DABAC

To discuss the expressive power of DABAC, we compare it with two recent ABAC modeling schemes whose goals overlap with our modeling scheme: (1.) versatility to express diverse ABAC policies and (2.) dynamic analysis capabilities. More specifically, we have selected $ABAC_\alpha$ (Jin et al., 2012) and C-ABAC (Fernández et al., 2019) (see § 2). The goal is to demonstrate that DABAC can be used to formalize instances of both modeling scheme. Due to space limitations, we will present an informal argumentation based on the most relevant formalisms; a more comprehensive study of both DABAC expressiveness and expressive power is subject to ongoing work.

$ABAC_\alpha$. Other than DABAC, $ABAC_\alpha$ does not generalize ABAC semantics, but instead aims at being as specific as possible. This results in model-independently predefined EN for subjects S , users U , and objects O . While $ABAC_\alpha$ do not explicitly define AV, they are implicitly part of attribution functions, which again reflect the three entity sets. To enable model-specific user attributes for example, there is $UA = \{ua \mid ua : U \rightarrow V_{ua}\}$, where V_{ua} is either an AV or a power set of an AV; subject- and object-attributes are defined analogously. So instead of defining AA_0 based on AV_0 , as we did in Tab. 1, $ABAC_\alpha$ models would require the opposite step order (still resulting in the same semantics). Only direct attribution is supported. Because of its deliberate choice of semantic

frugality, $ABAC_\alpha$ does not make a distinction between internal and external domains.

While the ACF of $ABAC_\alpha$ models can be flexibly defined in first-order logic, the function $Authorization_p : S \times O \rightarrow \mathbb{B}$ defined for this purpose differs from $auth_p$ functions in DABAC in that it requires to specify how attribute values for the entities passed as parameters (instead of AV elements) are derived. For DABAC, we deliberately deny this opportunity by the formal definition of AR in order to prevent possible conceptional errors introduced by specifying this part of the policy. The mindset here is that, for clean ABAC semantics, permissions have to be granted exclusively based on attribution, but never on identity – which is possible in $ABAC_\alpha$, again deliberately to support IBAC policies as well. Having that said, this difference relates to intended usage rather than expressive power, since any attribute values passed to an $auth_p$ can be derived in the VAR section of STS definitions in DABAC.

Although not in automaton notation, $ABAC_\alpha$ models are dynamic. Any model-specific STS however is fixed to a set of eight state-modifying operations, each with their predefined PRE and POST expressions, which might be rewritten in DABAC notation in a straight-forward way. This is a main reason for strong analytical features such as decidable safety (Ahmed and Sandhu, 2017), but also restricts $ABAC_\alpha$ expressive power to dynamics less general than in DABAC.

C-ABAC. C-ABAC takes a different approach than $ABAC_\alpha$, which shares more similarities with DABAC in terms of model engineering. This mainly originates from the generalized semantics of ABAC abstractions inspired from CBAC in both cases; it however results in different formalisms due to our requirements of automaton-based analysis and functional policy specification. Still, because of their generality, basic C-ABAC semantics can be transformed into DABAC in a straight-forward way.

From what C-ABAC calls entities, both its families \mathcal{P} (principals) and \mathcal{R} (resources) precisely map to EN for the internal domain. Since Env represents environment entities (such as networks, channels etc. (Fernández et al., 2019)), this set equals EN^{ext} . Another, abstract set of entities is based on CBAC *categories*: the set C in C-ABAC is an additional level of indirection between EN and AV, which can be used to express attribute classification and parameterization. In DABAC, this exact use case is covered by indirect attribution, so C is part of AV_1 here.

Two C-ABAC families \mathcal{At} and \mathcal{V} are used for model-specific definitions of attributes and AV, respectively: while the former defines an abstract, model-

specific set of attribute identifiers, the latter represents appropriate (model-specific) AV. The actual attribution, expressed by *att* functions in DABAC, is then formalized through four relations: \mathcal{PAtA} for AA_0 functions on EN in \mathcal{P} , \mathcal{RAtA} for AA_0 functions on EN in \mathcal{R} , and \mathcal{EAtA} for AA_0 functions on EN^{ext} , and \mathcal{CAAtA} for AA_1 functions. Each of these represents attribution by association with multiple tuples from $\mathcal{At} \times \mathcal{V}$.

The most distinct feature of C-ABAC in contrast to DABAC is how it handles operational semantics: While the DABAC requirement of functional specification and analysis resulted in an automaton model, C-ABAC follows CBAC in a purely axiomatic approach. This has two important consequences: (1.) The ACF of C-ABAC models has a more structured formalization than in DABAC. Instead of operation-specific conjunctions of $auth_p$ definitions, which a model engineer might mix and match without further restriction, C-ABAC defines a model-independent *authorization axiom*. While perfectly rewritable in the PRE sections of a DABAC STS, this normalization supports static analyses of C-ABAC models by stronger assumptions on authorization conditions. (2.) With respect to state-modification, the POST notation of DABAC goes beyond C-ABAC since it explicitly specifies model dynamics. As already discussed in § 2, DABAC does not separate administrative from non-administrative operations, for which the authors of C-ABAC presented Admin-CBAC as a parallel modeling scheme instead. Finally, it should be highlighted that in terms of *expressiveness* rather than *expressive power*, all three modeling schemes discussed here are tailored to their respective use cases – since mere simulation of common ABAC semantics does not necessarily mean that model formalisms are suitable for their intended purposes, such as analyzing static or dynamic properties or enabling tool-supported implementation. With an MSPE workflow and dynamic properties to analyze as a goal, we have tailored DABAC to support this specific use case without sacrificing the opportunity to switch to another modeling scheme as soon as such becomes more appropriate for practical reasons.

4 PRACTICAL APPLICATION

The key motivation for DABAC is to support MSPE by enabling comprehensive analyses of ABAC policies regarding dynamic model properties. In this section, we demonstrate the practical application of DABAC to an exemplary policy based on a scenario in the health care domain. Balancing security and usability in this scenario is practically challenging. On the one hand, confidentiality and integrity of health care data

as well as preventing unauthorized access to sensors and actuators is a critical requirement, on the other hand, patient-oriented treatment workflows must be possible at any time. In the following, we will guide through the steps of engineering a DABAC model for this policy (§ 4.1) and refining it based on a model analysis (§ 4.2) in a simplified MSPE process.

4.1 Model Engineering

The exemplary security policy and its DABAC model are settled in the context of a smart health care scenario comprising a hospital information system (HIS) for medical data management with remote patient monitoring capabilities based on external medical IoT sensors and actuators, such as can be found in (Ray et al., 2017; Mukherjee et al., 2017).

Primitive Model Components. Acting system entities are *users*, which are either human users or their personal (smart) devices. The following entities are subject to accesses by users:

- *objects*, representing data objects, such as patient-specific electronic health records (EHRs);
- *sensors*, representing interfaces to health monitoring sensors connected to patients, such as clinical thermometers;
- *actuators*, representing interfaces to medical actuators connected to patients, such as insulin pumps.

In model terms, user entities (set U) and object entities (set O) are internal entities. In contrast, both sensor entities (set Sen) and actuator entities (set Act) are logically and physically separate from the AC system, hence external entities. For illustration purposes, O represents EHRs, Sen represents thermometers, and Act represents insulin pumps. The individual elements of these sets are patient-specific, i. e. for each a one-to-one mapping to patients is assumed.

The considered policy is modeled based on attributes which reflect the hospital's organizational structure and workflows. Each user is assigned a *role*, such as physician, nurse, or patient, that enables accessing (passive) entities for the accomplishment of tasks. Orthogonally, each user is assigned to a hospital *ward*, such as internal, ICU, or surgery. Consequently, user entities are attributed by two functions att_{UR} and att_{UW} with role and ward attribute values (from R respectively W). For sake of simplicity, we allow one role and one ward per user.

Rather than a patient being purely assigned to a ward for the treatment of a disease and, thus, *all* users assigned to that ward having access to her EHR, the policy is specified more fine-grained: By introducing

- ▷ $auth_{read}(r_u \in R) \Leftrightarrow r_u \neq rPatient$
- ▷ $auth_{shareCases}(I_u \subseteq I, I_o \subseteq I) \Leftrightarrow I_u \cap I_o \neq \emptyset$
- ▷ $auth_{assign}(r_u \in R) \Leftrightarrow r_u = rManager \vee r_u = rPhysician$
- ▷ $auth_{isAssignee}(r_u \in R, r_{u'} \in R) \Leftrightarrow ((r_u = rManager \wedge r_{u'} = rPhysician) \vee r_{u'} = rNurse)$
- ▷ $auth_{delegate}(r_u \in R) \Leftrightarrow r_u = rPhysician$
- ▷ $auth_{isDelegate}(r_u \in R) \Leftrightarrow r_u = rPhysician$

Figure 1: Exemplary definitions of authorization functions for permissions in the DABAC HIS model.

treatment *cases*, only exactly those physicians and nurses assigned to care in a case will have access to an EHR. Additionally, assigned treatment personnel may belong to different wards. In our model, this is realized by the attribute value set I containing medical case identifiers (represented by indexing) as well as the attributions att_{UI} and att_{OI} , which attribute (1.) users related to that case (assigned physicians or nurses, patients treated) and (2.) EHR objects that represent the information about that case.

The external entities, thermometer sensors and insulin pump actuators, are attributed as follows: each temperature sensor is attributed with a temperature value from the set $Temp$ by att_{ST} and each insulin pump actuator is attributed with a dose value from the set $Dose$ by att_{AD} .

Following the components from EN, AV, and AA families, we now conclude by defining the AR families. Generally, AR contains a set of attribute-checking permissions (modeled as authorization functions). As an example, we have presented a selection of definitions in Fig. 1, which, subsequently, are also used in the STS operations. Their semantics are as follows:

$auth_{read}$: The permission read is granted any role other than rPatient.

$auth_{shareCases}$: The permission shareCases is granted iff given case attribute value sets I_u and I_o have a non-empty intersection.

$auth_{assign}$: The permission assign is granted for the roles rManager or rPhysician.

$auth_{isAssignee}$: The permission isAssignee is granted iff either a physician or a nurse is assignee, where only managers can assign cases to physicians.

$auth_{delegate}$: The permission delegate is granted for the delegator role rPhysician.

$auth_{isDelegate}$: The permission isDelegate is granted for the delegator role rPhysician.

Note that $auth_{delegate}$ and $auth_{isDelegate}$ have different semantics here, their identical definition is merely due to the simplifications of the example scenario.

Table 2: Overview of DABAC HIS primitive model components and their instantiation (functions written in relational style).

	Symbol	Description	HIS Instantiation
EN	U	set of user identifiers	{drCox, drKelso, drJD, nurseCarla, nurseLaverne, mrsFriendly, mrBruise, msPregnant, ...}
	O	set of (data) object identifiers	{ehrMrsFriendly, ehrMrBruise, ehrMsPregnant, ...}
EN^{ext}	Sen	set of sensor identifiers	{sThermometerMsPregnant, ...}
	Act	set of actuator identifiers	{actPumpMrsFriendly, ...}
AV_0	R	set of role attribute values	{rPhysician, rPatient, rNurse, rParamedic, rManager, rClerk}
	W	set of ward attribute values	{wInternal, wICU, wSurgery, wCardiology, wMaternity}
	I	set of numerical case identifiers	$I \subseteq \mathbb{N}$
AV_0^{ext}	$Temp$	set of temperature attribute values	$Temp \subseteq \mathbb{N}$
	$Dose$	set of drug dose attribute values	$Dose \subseteq \mathbb{R}$
AA_0	$att_{UR} : U \rightarrow R$	user-role-attribution	{(nurseCarla, rNurse), (drCox, rPhysician), (drKelso, rPhysician), ...}
	$att_{UW} : U \rightarrow W$	user-ward-attribution	{(nurseCarla, wInternal), (drCox, wInternal), (drKelso, wMaternity), ...}
	$att_{UI} : U \rightarrow \mathcal{P}(I)$	user-cases-attribution	{(drKelso, {42}), ...}
	$att_{OI} : O \rightarrow \mathcal{P}(I)$	EHR-cases-attribution	{(ehrMsPregnant, {42}), ...}
AA_0^{ext}	$att_{ST} : Sen \rightarrow Temp$	sensor-temperature-attribution	not applicable
	$att_{AD} : Act \rightarrow Dose$	actuator-dose-attribution	not applicable
AR	AR	set of permission authorization functions	{ $auth_{\text{read}}, auth_{\text{shareCases}}, auth_{\text{assign}}, auth_{\text{isAssignee}}, auth_{\text{delegate}}, auth_{\text{isDelegate}}, \dots$ }

All described primitive components and their instantiations (cf. automaton's initial state) for our model are summarized in Tab. 2.

Model Dynamics. In DABAC, model dynamics are defined based on a specialized deterministic automaton. The model dynamics are defined as described in § 3.2.2 and tailored model-specifically as follows:

- $\Gamma = \mathcal{P}(U) \times \mathcal{P}(O) \times \{att_{UR_\gamma} : U_\gamma \rightarrow R \mid U_\gamma \subseteq U\} \times \{att_{UW_\gamma} : U_\gamma \rightarrow W \mid U_\gamma \subseteq U\} \times \{att_{UI_\gamma} : U_\gamma \rightarrow \mathcal{P}(I) \mid U_\gamma \subseteq U\} \times \{att_{OI_\gamma} : O_\gamma \rightarrow \mathcal{P}(I) \mid O_\gamma \subseteq O\}$, where $\gamma = \langle U_\gamma, O_\gamma, att_{UR_\gamma}, att_{UW_\gamma}, att_{UI_\gamma}, att_{OI_\gamma} \rangle \in \Gamma$ is a single protection state;
- $OP = \{\text{readEHR}, \text{appendToEHR}, \text{createEHR}, \text{deleteEHR}, \text{fetchTemp}, \text{pushDose}, \text{addUser}, \text{removeUser}, \text{assignCase}, \text{delegateCase}, \text{revokeCase}\}$;
- Δ is defined by a set of operations according to the identifiers from OP . Generally, users may read or write objects, read but not write sensors, and write but not read actuators.² Fig. 2 illustrates three

²In practice, there might be devices such as ICU bedside monitors which both allow to monitor (“fetch”) real-time patient data, as well as to store (“push”) history records from newly arrived patients. These could be modeled by a common identifier, both in Sen and in Act . However, for the sake of simplicity, we assume $Sen \cap Act = \emptyset$ in this example.

- **readEHR**($u \in U_\gamma, o \in O_\gamma$) ::=
 - VAR: $r_u = att_{UR_\gamma}(u), I_u = att_{UI_\gamma}(u), I_o = att_{OI_\gamma}(o)$
 $w_u = att_{UW_\gamma}(u),$
 $I' = \bigcup_{u' \in U_\gamma \setminus \{u\}, att_{UW_\gamma}(u') = w_u} att_{UI_\gamma}(u')$
 - PRE: $auth_{\text{read}}(r_u) \wedge auth_{\text{shareCases}}(I_u, I_o) \wedge auth_{\text{shareCases}}(I', I_o)$
 - POST: true
- **assignCase**($u \in U_\gamma, u' \in U_\gamma, i \in I$) ::=
 - VAR: $r_u = att_{UR_\gamma}(u), r_{u'} = att_{UR_\gamma}(u'), I_u = att_{UI_\gamma}(u)$
 - PRE: $auth_{\text{assign}}(r_u) \wedge auth_{\text{isAssignee}}(r_u, r_{u'}) \wedge auth_{\text{shareCases}}(I_u, \{i\})$
 - POST: $att_{UI_\gamma'}(u') \leftarrow att_{UI_\gamma}(u') \cup \{i\}$
- **delegateCase**($u \in U_\gamma, u' \in U_\gamma, i \in I$) ::=
 - VAR: $r_u = att_{UR_\gamma}(u), r_{u'} = att_{UR_\gamma}(u'), I_u = att_{UI_\gamma}(u)$
 - PRE: $auth_{\text{delegate}}(r_u) \wedge auth_{\text{isDelegate}}(r_{u'}) \wedge auth_{\text{shareCases}}(I_u, \{i\})$
 - POST: $att_{UI_\gamma'}(u') \leftarrow att_{UI_\gamma}(u') \cup \{i\}$

Figure 2: Exemplary definitions of STS operations for the DABAC HIS model.

representative examples readEHR, assignCase, and delegateCase. Their semantics are as follows:

readEHR represents an operation to read an EHR. A user $u \in U_\gamma$ is allowed to read $o \in O_\gamma$ iff (1.) u is attributed a role $r_u \in R$ other than rPatient, i. e. no patient may read³ ($auth_{\text{read}}$), (2.) u shares cases

³With respect to the scenario's background, one might

Table 3: Overview of additions to DABAC HIS primitive model components and their instantiation.

	Symbol	Description	HIS Instantiation
AV_0, AV_1	T	set of team attribute values	$T \subseteq \mathbb{N}$
AA_0	$att_{UT} : \{u \in U \mid att_{UR}(u) \in \{rPhysician, rNurse, rParamedic, rManager\}\} \rightarrow T$	user-team-attribution	$\{\langle drKelso, 4242 \rangle, \langle drCox, 4242 \rangle, \langle nurseCarla, 4242 \rangle, \dots\}$
AA_1	$att_{TI} : T \rightarrow \mathcal{P}(I)$	team-case-attribution	$\{\langle 4242, \{42\} \rangle, \dots\}$

with o ($auth_{shareCases}$), and (3.) there is at least one user, who shares cases with o and who is from the reader’s own ward (last statement in VAR and $auth_{shareCases}$). Given all permission checks evaluate to true, POST does not have any effect on the model state and, thus, evaluates to true.

assignCase represents the assignment of treatment capabilities. A user $u \in U_\gamma$ is allowed to assign a case $i \in I$ to user $u' \in U_\gamma$ iff (1.) u is attributed with role $rPhysician$ or $rManager$, i. e. the assigner is either a physician or manager ($auth_{assign}$), (2.) the assignee u' is attributed with $rNurse$ or, if u is a manager, $rPhysician$ ($auth_{isAssignee}$), and (3.) u is attributed with case i ($auth_{shareCases}$). Given all permissions are successfully evaluated to true, POST renders a new state by additionally attributing u' .

delegateCase covers medical referrals and treatment delegations in the sense of “a physician assigns another physician responsibility for some case”. Thus, a user $u \in U_\gamma$ is allowed to delegate a case $i \in I$ to user $u' \in U_\gamma$ iff (1.) u is attributed with role $rPhysician$ ($auth_{delegate}$), (2.) the delegate u' is attributed with role $rPhysician$ as well ($auth_{isDelegate}$), and (3.) u is attributed with case i ($auth_{shareCases}$). Given all checks evaluate to true, POST renders a subsequent state by attributing u' with i .

4.2 Model Analysis

Once a model has been designed, the model engineer is interested in whether it meets the required correctness guarantees. Such questions are asked and answered in the model analysis step.

Analysis. We exemplarily consider the following practical analysis question in the given scenario: Is nurseCarla ever able to read ehrMsPregnant? This precisely reflects a query regarding the $\langle op \rangle$ -safety (Def. 5), where op equals readEHR. Based on the

argue that a patient should be allowed to read her own EHR at any time. This could be easily achieved by just removing the $auth_{read}$ authorization clause in PRE. Note that this relaxation cannot be made for appendToEHR.

given initial state γ_0 , this query would be expressed by $\langle \gamma_0, readEHR, [nurseCarla, ehrMsPregnant] \rangle$.

In general, safety properties are undecidable. Nevertheless to be able to perform safety analyses, a possibility is to restrict a model w. r. t. its dynamics and, thus, its expressive power (cf. ABAC $_\alpha$ and AdminCBAC). Since our goal is to maintain a model’s expressive power, we instead exploit the semi-decidability of safety: Given a model state γ and another state γ' reachable from γ , it is efficiently decidable whether in γ' the considered safety property is violated.

In order to find such states efficiently – especially in practical complex and extensive model instances – heuristic approaches have been developed. Sophisticated heuristics, such as DepSearch (Amthor et al., 2013) and WSDepSearch (Schlegel and Kühnhauser, 2020), are based on the dependency search approach exploiting inter-dependencies of STS operations, where the necessary pre-condition of an operation are established by the post-condition of another operation. Thus, in order to show the unsafety of a given model state, a privilege escalation is explicitly induced by executing a sequence of STS operations and a clever choice of operation parameters.

By leveraging a heuristic algorithm as outlined, a possible sequence of operations and inputs leading to a safety-violating state can be also identified in the given model instance and rendered as follows:

1. delegateCase(drKelso, drCox, 42);
2. assignCase(drCox, nurseCarla, 42);
3. readEHR(nurseCarla, ehrMsPregnant).

Interpretation. To enable nurseCarla to execute readEHR on ehrMsPregnant, the following pre-clauses must be met: (1.) she must be authorized to read – this is true since she is a nurse, not a patient; (2.) she must be assigned to at least one case that ehrMsPregnant is also assigned to; (3.) at least one user of the same ward must be assigned that case.

The latter two pre-clauses must have been explicitly established beforehand due to the assignments in the initial state (cf. Tab. 1). Being assigned to case 42 by drCox via assignCase satisfies the second pre-clauses for nurseCarla. For assignCase to be exe-

cutable (1.) drCox as assigner must be either physician or manager – this is true; (2.) drCox as assigner and physician must be attributed with this case; (3.) nurse-Carla must be a nurse – this is also true.

Both, the second pre-clauses of assignCase and third pre-clauses of readEHR, become satisfiable by the delegation of case 42 to drCox by drKelso via delegateCase: (1.) drKelso as delegator and drCox as delegate are both physicians; (2.) drKelso is assigned to this case. Consequently, a potential privilege escalation is possible on a ward-by-ward base, i. e. if at least one member (any one) of ward x is capable of accessing cases originally treated in ward y , this also holds for any other member of x . If, then again, a different member of x is capable of accessing cases from ward z , this may spread transitively.

Consequences. The model originally formalizes a policy which is based on the organizational structure of a hospital. Now, the analysis results produced a more substantial understanding of possible privilege escalation vulnerabilities that are anchored in this design. To address these design flaws, a reiteration step in the model engineering phase of MSPE is necessary to improve the model design.

One possible approach to fix the above-discussed privilege escalation phenomenon is summarized in Tab. 3. In order to cover the semantics of ward-crossing treatment workflows, the attribute *team*, the user-team-attribution *att_{UT}*, and the indirect team-case-attribution *att_{TI}* are introduced. A team consists of users, possibly from different wards, such that any team could commonly treat cases independent from their “original” wards.

5 CONCLUSIONS

This paper presents DABAC, a modeling scheme for ABAC policies that enables automaton-based analysis and specification of ABAC policies. By classifying formal abstractions in the semantical levels of primitive model components, model dynamics, and safety properties, we support a mix-and-match approach that streamlines the integration of DABAC models in the MSPE workflow. Based on a common ABAC use-case, we have highlighted how this approach could pave the way to a more standardized, tool-supported engineering of ABAC systems.

This is also reflected in our ongoing work: supporting DABAC-based MSPE by tools in two areas of application. These are (1.) semi-automated model implementation, based on our previous work on model specification languages (Amthor and Schlegel, 2020);

(2.) real-time monitoring of model invariants which cannot be proven during model analysis, such as is the case with a heuristically analyzed but formally undecidable safety definition. For a real-world application integration, our future work aims at leveraging DABAC to implement correct and reliable risk-based ABAC policies based on real-time threat information from external sources.

REFERENCES

- Ahmed, T. and Sandhu, R. (2017). Safety of ABAC $_{\alpha}$ is Decidable. In *NSS '17*, pages 257–272.
- Amthor, P. (2017). Efficient Heuristic Safety Analysis of Core-based Security Policies. In *SECURITY '17*, pages 384–392.
- Amthor, P. (2019). *Aspect-oriented Security Engineering*. ISBN 978-3-7369-9980-0.
- Amthor, P., Kühnhauser, W. E., and Pölck, A. (2013). Heuristic Safety Analysis of Access Control Models. In *SACMAT '13*, 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 Schlegel, M. (2020). Towards Language Support for Model-based Security Policy Engineering. In *SECURITY '20*, pages 513–521.
- Barker, S. (2009). The Next 700 Access Control Models or a Unifying Meta-Model? In *SACMAT '09*, pages 187–196.
- Basin, D., Clavel, M., and Egea, M. (2011). A Decade of Model-Driven Security. In *SACMAT '11*, pages 1–10.
- Bertolissi, C., Fernández, M., and Thuraisingham, B. (2020). Admin-CBAC: An Administration Model for Category-Based Access Control. In *CODASPY '20*, pages 73–84.
- Bhatt, S. and Sandhu, R. (2020). ABAC-CC: Attribute-Based Access Control and Communication Control for Internet of Things. In *SACMAT '20*, pages 203–212.
- Biswas, P., Sandhu, R., and Krishnan, R. (2016). Label-Based Access Control: An ABAC Model with Enumerated Authorization Policy. In *ABAC '16*, pages 1–12.
- Fernández, M., Mackie, I., and Thuraisingham, B. (2019). Specification and Analysis of ABAC Policies via the Category-Based Metamodel. In *CODASPY '19*, pages 173–184.
- Ferraiolo, D., Chandramouli, R., Kuhn, R., and Hu, V. (2016). Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC). In *ABAC '16*, pages 13–24.
- Gupta, M., M. Awaysheh, F., Benson, J., Alazab, M., Patwa, F., and Sandhu, R. (2020). An Attribute-Based Access Control for Cloud-Enabled Industrial Smart Vehicles. *III*, 17(6):4288–4297.
- Harrison, M. A., Ruzzo, W. L., and Ullman, J. D. (1976). Protection in Operating Systems. *Comm. of the ACM*, 19(8):461–471.

- Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., and Scarfon, K. (2014). Guide to Attribute Based Access Control (ABAC) Definition and Considerations. NIST Special Publication 800-162.
- Jha, S., Sural, S., Atluri, V., and Vaidya, J. (2019). Security Analysis of ABAC under an Administrative Model. *IET Inf. Secur.*, 13(2):96–103.
- Jin, X., Krishnan, R., and Sandhu, R. (2012). A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In *DBSec '12*, volume 7371, pages 41–55.
- Lang, B., Foster, I. T., Siebenlist, F., Ananthkrishnan, R., and Freeman, T. (2009). A Flexible Attribute Based Access Control Method for Grid Computing. *J. Grid Comput.*, 7(2):169–180.
- Marin, M., Kutsia, T., and Dundua, B. (2019). A Rule-based Approach to the Decidability of Safety of ABAC α . In *SACMAT '19*, pages 173–178.
- Mukherjee, S., Ray, I., Ray, I., Shirazi, H., Ong, T., and Kahn, M. G. (2017). Attribute Based Access Control for Healthcare Resources. In *ABAC '17*, pages 29–40.
- Narouei, M., Khanpour, H., Takabi, H., Parde, N., and Nielsen, R. (2017). Towards a Top-down Policy Engineering Framework for Attribute-based Access Control. In *SACMAT '17*, pages 103–114.
- Ray, I., Alangot, B., Nair, S., and Achuthan, K. (2017). Using Attribute-Based Access Control for Remote Healthcare Monitoring. In *SDS '17*, pages 137–142.
- Schlegel, M. and Amthor, P. (2020). Beyond Administration: A Modeling Scheme Supporting the Dynamic Analysis of Role-based Access Control Policies. In *SECRYPT '20*, pages 431–442.
- Schlegel, M. and Kühnhauser, W. (2020). Exploiting Hot Spots in Heuristic Safety Analysis of Dynamic Access Control Models. In *SECRYPT '20*, pages 522–532.
- Stoller, S. D., Yang, P., Gofman, M., and Ramakrishnan, C. R. (2009). Symbolic Reachability Analysis for Parameterized Administrative Role Based Access Control. In *SACMAT '09*, pages 165–174.
- Tripunitara, M. V. and Li, N. (2007). A Theory for Comparing the Expressive Power of Access Control Models. *J. Comput. Secur.*, 15(2):231–272.
- Tripunitara, M. V. and Li, N. (2013). The Foundational Work of Harrison-Ruzzo-Ullman Revisited. *TDSC*, 10(1):28–39.
- Vimercati, S. D. C. d., Samarati, P., and Jajodia, S. (2005). Policies, Models, and Languages for Access Control. In *DNIS '05*, volume 3433/2005 of *LNCS*, pages 225–237.
- Xu, Z. and Stoller, S. D. (2015). Mining Attribute-Based Access Control Policies. *TDSC*, 12(5):533–545.