Reduction of Access Control Decisions

Charles Morisset
Centre for Cybercrime and Computer Security
Newcastle University, U.K.
charles.morisset@newcastle.ac.uk

Nicola Zannone
Eindhoven University of Technology
n.zannone@tue.nl

ABSTRACT

Access control has been proposed as "the" solution to prevent unauthorized accesses to sensitive system resources. Historically, access control models use a two-valued decision set to indicate whether an access should be granted or denied. Many access control models have extended the two-valued decision set to indicate, for instance, whether a policy is applicable to an access query or an error occurred during policy evaluation. Decision sets are often coupled with operators for combining decisions from multiple applicable policies. Although a larger decision set is more expressive, it may be necessary to reduce it to a smaller set in order to simplify the complexity of decision making or enable comparison between access control models. Moreover, some access control mechanisms like XACML v3 uses more than one decision set. The projection from one decision set to the other may result in a loss of accuracy, which can affect the final access decision. In this paper, we present a formal framework for the analysis and comparison of decision sets centered on the notion of decision reduction. In particular, we introduce the notion of safe reduction, which ensures that a reduction can be performed at any level of policy composition without changing the final decision. We demonstrate the framework by analyzing XACML v3 against the notion of safe reduction. From this analysis, we draw guidelines for the selection of the minimal decision set with respect to a given set of combining operators.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access controls

General Terms

Security, Theory

Keywords

Policy evaluation, access decision, XACML, formal analysis

SACMAT'14, June 25–27, 2014, London, Ontario, Canada.

http://dx.doi.org/10.1145/2613087.2613106.

1. INTRODUCTION

An access control mechanism can generally be seen as a system taking as input an access query from the environment (typically submitted by a user) and deciding whether to allow this query or not, possibly by collecting the current contextual information. Although such a mechanism can also decide to modify the query to make it acceptable, or to request some obligations to be fulfilled at runtime, it should eventually decide whether to allow the query or not.

An access control mechanism usually relies on one or several *access control policies*, which are functions associating access queries with decisions. In other words, given a set of queries $\mathcal{Q}$ and a set of decisions $\mathcal{D}$, a policy is a function $\pi : \mathcal{Q} \to \mathcal{D}$. Whenever the range of a policy π is a subset of $\{1, 0\}$, indicating that a query should be allowed and denied, respectively, we say π is *enforceable*, i.e., an access control mechanism can directly enforce this policy without the need of interpreting the decision. For instance, a trivial example of enforceable policy is a function π_1 such that for any query q, $\pi_1(q) = 1$, i.e., the policy allowing every possible query.

However, defining a single enforceable policy for a large set of queries can be quite challenging: a simple enumeration of the allowed (or denied) queries is likely to contain mistakes, especially when the system changes. Hence, the usual approach to define an enforceable policy is compositional, i.e., sub-policies corresponding to different concerns are first defined, and then composed together. Each sub-policy can also be defined in a compositional way. For instance, a security designer might want to define: a confidentiality policy over read accesses and an integrity policy over write accesses; a different policy for each business role in a company; a regular policy and an emergency policy, and so on.

It is therefore often useful to introduce decisions beyond 1 and 0 in order to represent the possible interactions between the different policies. For instance, a typical decision is "not-applicable" (denoted here $\perp$), which indicates that a policy cannot be applied to a query. In the previous examples, a policy regulating the read accesses to a file is not applicable to queries for write access, and thus it cannot make any *conclusive* decision for such queries. Similarly, the policy for a nurse might not be applicable to a physician or an administrator. We present in Section 2 several examples of decision sets, which usually come with specific decision operators, describing how to compose decisions.

Although a larger set of decisions provides more flexibility to compose sub-policies, it also implies that the access control mechanism might not know how to interpret decisions which are neither 1 nor 0. For instance, a mechanism

having to control a query q for which the policy returned $\perp$ is facing two options: it could deny q, interpreting $\perp$ as "providing no evidence that the query should be granted", or allow q, interpreting $\perp$ as "providing no evidence that the query should be denied". In addition, different access control mechanisms might need to interact together in order to provide the final decision, and if they use different decision sets, it is not directly possible to combine their decisions.

It is therefore useful, when introducing new decisions, to also introduce a *reduction* function, which defines how to reduce a decision set to a smaller one. Such a function serves several purposes: it allows the mechanism to make a conclusive decision by reducing inconclusive ones; it allows to reuse on a smaller set of decisions operators defined on a larger set; it provides a common ground for different mechanisms to interact. A reduction function between decision sets, however, can introduce weaknesses in the access control mechanism. Indeed, it might change the intended semantics of a policy. In particular, applying a reduction function to different stages of the policy evaluation process may result in a different decision. These considerations raise the main research question addressed in this paper: *How (and when) to reduce a decision set while preserving the policy semantics?*

The main contribution of this paper is a formal framework around the notion of reduction between decision sets. In particular, we introduce the notion of *safe reduction*, which ensures that a reduction can be performed at any level of policy composition without changing the final decision. We illustrate the framework by analyzing XACML v3 [2], the de facto standard for the specification and enforcement of access control policies. What makes XACML v3 an interesting access control mechanism to investigate is that it uses two decision sets. In particular, it uses the four-valued decision set of XACML v2 [1] in which indeterminacy is represented using a single decision value; in addition, it introduces an extended set for indeterminacy to represent the potential decision to be returned if there would not have been an error causing the indeterminacy. Combining operators in XACML v3 are defined over either a decision set or the other. Therefore, decision values have to be projected from one decision set to the other depending on the used operator. We show that the reduction between these two sets defined in [2] may lead to counter-intuitive results. Based on this analysis, we provide guidelines for the selection of the minimal decision set preserving the policy semantics with respect to a given set of combining operators.

The remainder of the paper is organized as follows. The next section provides an overview of the decision sets defined in the literature. Section 3 introduces the notion of decision reduction and discusses the issues that can arise when a decision set is reduced using some examples. Section 4 formalizes the notion of safe reduction. Section 5 presents an analysis of XACML v3 against the notion of safe reduction and provides guidelines for the selection of the minimal decision set. Finally, Section 6 concludes the paper.

2. DECISION SET

To the best of our knowledge, there is no related work specifically on the reduction of one decision set to another. There is, however, a rich literature on the definition of different decision sets, in order to represent different problems. In this section, we present an overview of some of these decision sets. In general, a decision set is a set $\mathcal{D} = \{d_1, \ldots, d_n\}$, where each d_i represents a distinct decision that can be returned by a policy. For the sake of clarity, we refer to 1 and 0, representing allow and deny respectively, as conclusive decisions, since they can be directly enforced by an access control mechanism.

2.1 Two-valued

Historically, the decision set was simply $\mathcal{D}_2 = \{0, 1\}$. For instance, Lampson [15] introduces the notion of access matrix, where a subject can access an object if and only if the corresponding access right exists in the matrix. Similarly, the protection matrix of the HRU model [13] explicitly denies or allows accesses. Bell-Lapadula [5] and Chinese Wall [6] models extends the notion of access matrix by considering a lattice of levels of security and conflict of interest classes, respectively, but still define a strict partition of accesses between those allowed and those denied. Similarly, the RBAC model [12] introduces more flexibility with the notion of role, but an access is allowed if the subject has a role with the corresponding decision, and denied otherwise.

The classical Boolean operators can be used when composing access control policies ranging in $\mathcal{D}_2$.

2.2 Three-valued

Specifying and managing access control policies is a difficult and error-prone task which may lead to grant permissions to the wrong users. To this end, policy languages should allow the specification of exceptions to make it explicit in which cases access should not be granted. The concept of exception has led to the differentiation between positive and negative policies [11], where positive policies are used to represent privileges and negative policies are used to represent restrictions. This idea has been further extended by the introduction of structured languages to define security policies [21]. Although this allows the specification of fine-grained access control policies and facilitates policy management, it introduces some challenges.

An access control policy unlikely covers all possible queries that can be specified. This results in situations in which any (positive or negative) policy that matches a given query does not exist. In addition, a query may be *incomplete*, i.e., there is not enough available information to determine whether an access should be granted or denied [21]. To capture these situations, it is necessary to introduce a decision that indicates that a policy is not applicable.

On the other hand, the adoption of structured languages allows the specification of composite policies. Intuitively, a policy can be formed of sub-policies, where each sub-policy corresponds to a different concern of the protection system or represent the same concerns from the perspective of a different authority. In this setting, more than one sub-policy may be applicable to a given query. Therefore, it is necessary to have mechanisms to combine the decisions returned by different applicable policies.

Crampton and Huth [9] consider the set $\mathcal{D}_3 = \{1, 0, \perp\}$, where $\perp$ stands for the non-applicable decision. To combine decisions obtained by applying different applicable policies, they provide a characterization of combining operators over $\mathcal{D}_3$. A binary operator $\oplus$ is said to be a $\cup$-operator when $x \oplus \perp = x = \perp \oplus x$, for any decision x. Similarly, an operator $\oplus$ is said to be a $\cap$-operator when $x \oplus \perp = \perp = \perp \oplus x$, for any x. Rao et al. [18] define the Fine-grained Integration Algebra, which supports policies ranging in $\mathcal{D}_3$. Furthermore,

$\sqcap$	1	0	$\bot$
1	1	0	$\bot$
0	0	0	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

(a) Weak conjunction

$\sqcup$	1	0	$\bot$
1	1	1	$\bot$
0	1	0	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

(b) Weak disjunction

$\tilde{\sqcap}$	1	0	$\bot$
1	1	0	$\bot$
0	0	0	0
$\bot$	$\bot$	0	$\bot$

(c) Strong conjunction

$\tilde{\sqcup}$	1	0	$\bot$
1	1	1	1
0	1	0	$\bot$
$\bot$	1	$\bot$	$\bot$

(d) Strong disjunction

Figure 1: Binary operators over $\mathcal{D}_3 = \{1, 0, \bot\}$

X	$\neg X$	$\sim X$
1	0	1
0	1	0
$\bot$	$\bot$	0

Figure 2: Unary operators over $\mathcal{D}_3 = \{1, 0, \bot\}$

they prove that their algebra is complete, i.e., it can express all possible ways of integrating policies. The language EPAL [4], an XML-based language for privacy policies, also uses non-applicable as third value.

Crampton and Morisset [10] use a three-valued logics for the evaluation of policy targets in PTaCL, based on Kleene algebra. Figures 1 and 2 define some operators for the three-valued logic over $\mathcal{D}_3$: $\sqcap$ and $\sqcup$ represent the weak conjunction and disjunction, respectively; $\tilde{\sqcap}$ and $\tilde{\sqcup}$ represent the strong conjunction and disjunction, respectively; $\neg$ is the negation operator; and $\sim$ is a "weakening" operator, transforming $\bot$ into 0. Jobe [14] proved that the set of operators $\{\tilde{\sqcup}, \neg, \sim\}$ is functionally complete.

2.3 Four-valued

Many factors may influence an access decision; capturing these factors in the decision requires extending the decision set. Two main factors have led to the extension of $\mathcal{D}_3$ with an additional value: (i) the notion of conflict and (ii) errors which can occur in policy evaluation.

2.3.1 Conflict

As said above, traditional access control models, such as those described in Section 2.1 only allow the specification of "positive" policies (i.e., policies that specify the actions that a user can perform on an object). Intuitively, an access is denied unless there exists a policy which matches the access query.

In contrast, structured languages allow the specification of composite policies comprising both positive and negative sub-policies to regulate the access to sensitive resources and data. This may result in conflicts during policy evaluation, as some policies can grant permission and others deny the very same permission.

Many policy algebras especially rooted on $\mathcal{D}_3$ aim to resolve conflicts, i.e., choosing between 1 or 0 when both decisions are returned by different sub-policies, for instance using the conjunction or disjunction operator. However, in some cases it is desirable to record that a conflict occurred. To this end, a more fine-grained decision set is needed. For instance, Bruns and Huth [7] introduce a new decision, denoted as $\top$, to allow for an explicit notion of conflict.

The new decision set $\mathcal{D}_4 = \{1, 0, \bot, \top\}$ can be intuitively obtained by considering the powerset of the two-valued decision set. In particular, the powerset of $\mathcal{D}_2$ is defined as $\wp(\mathcal{D}_2) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$. The meaning of each of these decisions can be understood as follows:

- $\emptyset$ indicates that the decision is neither 0 nor 1. In other words, the policy cannot reach any conclusive decision. This is modeled as the decision $\bot$ in [7], and can be interpreted as the non-applicable decision.
- $\{0\}$ and $\{1\}$ indicates that the policy reached the conclusive decision 0 and 1, respectively.
- $\{0, 1\}$ indicates that the policy reached the decision that the query should be both allowed and denied. This is modeled as decision $\top$ in [7], and can be interpreted as the conflict decision.

Intuitively, a decision in $\mathcal{D}_4$ corresponds to *all* the decisions in the set, and therefore the operators defined for $\mathcal{D}_2$ can be extended to $\mathcal{D}_4$ in a point-wise way, i.e., given any operator $op : \mathcal{D}_2 \times \mathcal{D}_2 \to \mathcal{D}_2$ and any sets $X, Y \subseteq \mathcal{D}_2$, we define operator $\overline{op} : \mathcal{D}_4 \times \mathcal{D}_4 \to \mathcal{D}_4$ as

$$\overline{op}(X, Y) = \{op(x, y) \mid x \in X \wedge y \in Y\}$$

According to [20], a language ensuring that no policy can return $\emptyset$ is said to be total, while a language ensuring that no policy can return $\{0, 1\}$ is said to be deterministic.

2.3.2 Indeterminate

Conflicts are not the only case where a conclusive decision cannot be made. Errors occurring in policy evaluation also introduce uncertainty into decision-making. A policy language which introduces a decision to indicate the occurrence of errors in the policy evaluation process is XACML v2 [1]. In particular, XACML v2 uses decision set {Permit, Deny, NA, Indeterminate}, where Permit represents the permit decision, Deny the deny decision, NA represents that the policy is not applicable to the given query, and Indeterminate indicates that the policy decision point is unable to evaluate the query because of some error (e.g., missing attributes, division by zero, network errors while retrieving policies, syntax errors in the query or in the policy). This decision set corresponds to $\mathcal{D}_4 = \{1, 0, \bot, \top\}$, where $\top$ now stands for the decision representing indeterminacy. XACML v2 also provides a number of combining algorithms to make a decision when more than one policy is applicable: permit-overrides, deny-overrides, first-applicable, only-one-applicable. Intuitively, these algorithms define procedures to evaluate composite policies based on the order of the sub-policies and priorities between decisions.

It is worth mentioning that even though, strictly speaking, XACML v2 considers a four-valued decision set, some combining algorithms distinguish the case where Indeterminate is obtained from a permit rule and the case where Indeterminate is obtained from a deny rule; however, such a distinction is not explicit. In other words, the semantics of decision Indeterminate might change according to the effect of the policy that returned it. As we show in the next section, XACML v3 clarifies this semantics by introducing the extended set for the Indeterminate decision.

Li et al. formalize in [16] the notion of policy combining algorithms as defined in XACML v2, and as such also considers the set $\mathcal{D}_4 = \{1, 0, \bot, \top\}$, where $\top$ stands for the Indeterminate decision. In addition to the XACML operators, they also define consensus and majority operators, using linear constraints. Finally, Arieli and Avron define in [3] the operators $\neg$, $\oplus$ and $\supset$, intuitively corresponding to the negation, non-deterministic choice and implication, respectively, and prove that $\{\neg, \oplus, \supset, \bot, \top\}$ is functionally complete for $\mathcal{D}_4$.

2.4 Six-valued

XACML v3 [2] uses two decision sets. Besides the decision set of XACML v2, XACML v3 refines the notion of indeterminacy by introducing the Indeterminate extended set: Indeterminate{P}, Indeterminate{D} and Indeterminate{PD}. Intuitively, the new Indeterminate decisions indicate the evaluation of a policy if an error would not have occurred. This results in the set $\mathcal{D}_6 = \{$Permit, Deny, NA, Indeterminate{P}, Indeterminate{D}, Indeterminate{PD}$\}$. The choice of the decision set depends on the algorithm used to combine policies. In particular, permit-overrides and deny-overrides are defined using the Indeterminate extended set and therefore over $\mathcal{D}_6$; in contrast, the other combining algorithms – first-applicable, permit-unless-deny, deny-unless-deny and only-one-applicable – are defined over $\mathcal{D}_4$.

2.5 Seven-valued

The intuition behind the extended set defined in XACML v3 is that an indeterminate decision effectively corresponds to several decisions. For instance, Indeterminate{P} indicates that a policy could have either evaluated to NA or to Permit, but not to Deny.

The language PTaCL [10] makes this intuition explicit by considering $\mathcal{D}_7 = \wp(\mathcal{D}_3)\backslash\emptyset$ as the decision set, i.e., a decision is a non-empty subset of $\mathcal{D}_3 = \{1, 0, \bot\}$, representing all the possible decisions. In this case, Indeterminate{P} is encoded as $\{1, \bot\}$. In a similar way than operators over $\mathcal{D}_2$ can be extended in a point-wise way to operators over $\mathcal{D}_4$, operators over $\mathcal{D}_3$ can be extended over $\mathcal{D}_7$. For instance, Figure 4 presents the definition of weak conjunction $\sqcap$ over $\mathcal{D}_7$.

2.6 Other decision sets

A decision set can also be defined in a more general way. For instance, the D-algebra [17] uses multi-valued logics, such as the Łukasiewicz logic, to represent the decision set. The D-algebra is defined for a non-empty set of decisions, and the operators, which intuitively correspond to negation, strong disjunction and test for equality, are algebraically specified.

A decision can also be defined as a risk value [8], i.e., as the probability of an event to happen if this decision is enforced multiplied by the impact of this event.

In general, there is no strict limit to the way a decision set can be defined. The definition of the decision set depends on the specific needs of the system in which the access control mechanism is deployed.

3. DECISION REDUCTION

As illustrated in the previous section, there is a large variety of access decision sets in the literature, typically introduced to address a specific problem to be addressed by the corresponding access control language. For instance, a language handling uncertainty in the environment might use the XACML decision Indeterminate, in order to represent that something unexpected occurred, or a language requiring an explicit notion of conflict might use the decision $\top$, as defined in PBel [7].

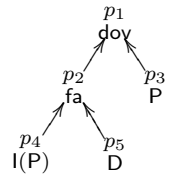
However, this multiplicity of decision sets raises several issues. First, it is not always clear how non-conclusive decisions (i.e., neither 1 nor 0) should be treated at the enforcement point. In addition, languages can evolve over time, adding new decisions to handle new aspects (e.g., XACML from v2 to v3), and it could be desirable to ensure that the operators defined over the new set are compatible with the former set. Finally, access control systems using different decision sets might need to interact in order to make a decision, in which case they need to have a common ground. For instance, systems of systems (SoS) are dynamic coalitions of distributed, autonomous and heterogeneous systems that collaborate to achieve a common goal, and each system in the SoS can use a different access control model along with a different decision set [19].

In order to address these issues, we introduce here the notion of *decision reduction*, such that, intuitively speaking, a decision set is reduced into a smaller one. We focus here on the size of the decision sets, making the implicit assumption that two decision sets with the same size are equivalent. This is clearly an over-simplification, and it can be argued that the two different four-valued sets presented in Section 2.3 associate a different meaning with the fourth value, one being for conflict, the other for indeterminacy. However, we focus here on operators, which are defined similarly on both sets. A rigorous approach would be to define a notion of abstract set, for instance $\mathcal{D}_4 = \{0, 1, 2, 3\}$, and then to consider different interpretations, mapping each abstract decisions to a concrete one, for instance $\mathcal{D}_{\text{XACML}} = \{0 \mapsto$ Deny, $1 \mapsto$ Permit, $2 \mapsto$ NA, $3 \mapsto$ Indeterminate$\}$ for the decision set of XACML v2. We believe there is little to be gained by adopting a rigorous approach here, and for the sake of simplicity, we do not make the distinction between abstract and concrete sets whenever it is clear from the context. In particular, we assume that the decision set of XACML v2, $\mathcal{D}_4$, is included into that of XACML v3, $\mathcal{D}_6$, such that Indeterminate and Indeterminate{PD} are just different interpretations of the same abstract decision. We argue that this assumption is reasonable: the XACML specification states that "the output of a combining algorithm which does not track the extended set of Indeterminate values MUST be treated as Indeterminate{PD} for the value Indeterminate by a combining algorithm which tracks the extended set of Indeterminate values" [2, C.1].

Hence, we now define the notion of reduction, which maps all decisions of a set to decisions of a subset, while leaving the decisions in the subset unchanged.

Definition 1 (Decision Reduction). *Given two decision sets $\mathcal{D}_n$ and $\mathcal{D}_m$ such that $\mathcal{D}_m \subseteq \mathcal{D}_n$, a reduction is a function $\rho : \mathcal{D}_n \to \mathcal{D}_m$ such that for any $d \in \mathcal{D}_m$, $\rho(d) = d$.*

Given two decision sets, more than one reduction function can be defined, corresponding to different interpretations of the reduced decisions. For instance, when reducing from $\mathcal{D}_3$ to $\mathcal{D}_2$, the decision $\bot$ can be projected either to 1 or to 0, corresponding to a less restrictive or a more conservative approach, respectively. Accordingly, we have re-



(a) Policy Fragment

fa first-applicable
dov deny-overrides

D Deny
P Permit
I(P) Indeterminate{P}

(b) Notation

Figure 3: Policy Example

ductions ρ^1_{32} and ρ^0_{32} where $\rho^1_{32}(\perp) = 1$, $\rho^0_{32}(\perp) = 0$ and $\rho^1_{32}(d) = \rho^0_{32}(d) = d$ for $d \in \{0, 1\}$.

A reduction from one set to another, however, may result in an unnecessary loss of accuracy, which has an effect on the final access decision. In particular, it might lead to accept a query that would have been intuitively denied, or conversely. In the remainder of the section, we discuss some examples in which decision reductions have an impact on the final access decision using some sample reductions.

XACML v3.

As mentioned in Section 2.4, XACML v3 is defined over two decision sets: $\mathcal{D}_4$ in which indeterminacy is modeled as a single decision Indeterminate, and $\mathcal{D}_6$ in which indeterminacy is modeled using the Indeterminate extended set. The Indeterminate extended set is only used for combining algorithms permit-overrides and deny-overrides. The other combining algorithms are defined over $\mathcal{D}_4$. Intuitively, decisions Indeterminate{P}, Indeterminate{D} and Indeterminate{PD} are projected onto decision Indeterminate when the policy has to be evaluated with respect to first-applicable, permit-unless-deny, deny-unless-permit, only-one-applicable. As said above, Indeterminate is considered equivalent to Indeterminate{PD} when the policy has to be evaluated with respect to permit-overrides and deny-overrides.

Consider the policy fragment in Figure 3. Policy p_1 consists of policies p_2 and p_3 which are combined using deny-overrides (dov, see Section 5 for its full definition). In turns, p_2 consists of policies p_4 and p_5 which are combined using first-applicable (fa). Suppose now that p_3 is evaluated to Permit, p_4 to Indeterminate{P}, and p_5 to Deny. Because of first-applicable, XACML v3 projects the decision of p_4 to Indeterminate, and thus $p_2 = \mathsf{fa}(p_4, p_5)$ evaluates to Indeterminate. The decision of p_2 is then projected to Indeterminate{PD}, and thus $p_1 = \mathsf{dov}(p_2, p_3)$ evaluates to Indeterminate{PD}. Now assume that the Indeterminate extended set is used with every combining algorithm. For instance, first-applicable can be intuitively extended over $\mathcal{D}_6$ by returning the decision of the first applicable policy (see Section 5 for a definition of first-applicable over $\mathcal{D}_6$). In this case, $p_2 = \mathsf{fa}(p_4, p_5)$ evaluates to Indeterminate{P}. Hence, $p_1 = \mathsf{dov}(p_2, p_3)$ evaluates to Permit.

From XACML v3 to XACML v2.

As another example of the loss of accuracy introduced by the extended set of XACML v3, consider the behavior of the operator dov with respect to the Indeterminate values, depending on whether evaluated in the context of XACML v2 or XACML v3. XACML v2 employs two versions of dov, one for combining rules and one for combining policies [1, C.1]. The operator dov for policy is based on a single interpretation of Indeterminate and has the following behavior:

$$\mathsf{dov}(\mathsf{Indeterminate}, \mathsf{Permit}) = \mathsf{Deny}$$

In contrast, dov for rules takes into account the effect of the rule in which the error occurred in order to make a decision. In particular, it has the following behavior:

$$\mathsf{dov}(\mathsf{Indeterminate}, \mathsf{Permit}) = \mathsf{Permit} \quad \text{if } e(I) = \mathsf{Permit}$$
$$\mathsf{dov}(\mathsf{Indeterminate}, \mathsf{Permit}) = \mathsf{Indeterminate} \quad \text{if } e(I) = \mathsf{Deny}$$

where $e(I)$ indicates the effect of the rule which returns Indeterminate.

XACML v3 uses a single operator dov for both rules and policies [2, C.2], which generalizes dov for rules of XACML v2. Formally,

$$\mathsf{dov}(\mathsf{Indeterminate\{P\}}, \mathsf{Permit}) = \mathsf{Permit}$$
$$\mathsf{dov}(\mathsf{Indeterminate\{D\}}, \mathsf{Permit}) = \mathsf{Indeterminate\{PD\}}$$
$$\mathsf{dov}(\mathsf{Indeterminate\{PD\}}, \mathsf{Permit}) = \mathsf{Indeterminate\{PD\}}$$

Evaluating dov in the context of XACML v2 or XACML v3 poses some issues. It is evident that the semantics of dov for policy in XACML v2 is different from the one of dov in XACML v3. This difference has a significant impact on policy evaluation. Suppose we need to evaluate expression $\mathsf{dov}(\mathsf{Indeterminate\{P\}}, \mathsf{Permit})$ in the context of XACML v2, where the value Indeterminate{P} needs to be reduced. We obtain different decisions depending on the point at which we apply the reduction. In particular, if we apply the reduction at the top level (i.e., first decisions are combined using dov of XACML v3 and then the decision is reduced), we obtain Permit; on the other hand, if we apply it at the subpolicy level (i.e., first decisions are reduced and then they are combined using dov of XACML v2), we obtain Deny (when dov for policy is used). Although we cannot necessarily decide which behavior is the correct one, we can analyze the operator dov to detect such inconsistencies, through the notion of *safe reduction*, introduced in Section 4.

From PTaCL to XACML v3.

PTaCL uses decision set $\mathcal{D}_7 = \wp(\{1, 0, \perp\}) \setminus \{\emptyset\}$, while XACML v3 uses decision set $\mathcal{D}_6 = \{\mathsf{Permit}, \mathsf{Deny}, \mathsf{NA}, \mathsf{Indeterminate\{P\}}, \mathsf{Indeterminate\{D\}}, \mathsf{Indeterminate\{PD\}}\}$. To reduce $\mathcal{D}_7$ to $\mathcal{D}_6$, one needs to project two decisions in $\mathcal{D}_7$ onto one decision in $\mathcal{D}_6$. Intuitively, the 3-valued logic $\{1, 0, \perp\}$ is used to represent Permit, Deny and NA, respectively. The notion of indeterminacy is modeled in PTaCL by considering *sets of decisions*, instead of single decisions. Thus, we have the following reduction $\rho^{\mathsf{I(PD)}}_{76}$:

$$\{1\} = \mathsf{Permit} \qquad \{1, \perp\} = \mathsf{Indeterminate\{P\}}$$
$$\{0\} = \mathsf{Deny} \qquad \{0, \perp\} = \mathsf{Indeterminate\{D\}}$$
$$\{\perp\} = \mathsf{NA} \qquad \{1, 0\}, \{1, 0, \perp\} = \mathsf{Indeterminate\{PD\}}$$

We now show that such a reduction may have an undesirable behavior with respect to operators used to combine decisions. Consider, for instance, the weak conjunction operator $\sqcap$ in Figure 1a extended to $\mathcal{D}_7$ in a point-wise way (Section 2.3). Figure 4 shows the encoding of weak conjunction in PTaCL.

An encoding of weak conjunction in XACML v3 (Figure 5) can be obtained by applying the reduction presented above

$\sqcap$	1	0	$\bot$	$1,\bot$	$0,\bot$	$1,0$	$1,0,\bot$
1	1	0	$\bot$	$1,\bot$	$0,\bot$	$1,0$	$1,0,\bot$
0	0	0	$\bot$	$0,\bot$	$0,\bot$	0	$0,\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$1,\bot$	$1,\bot$	$0,\bot$	$\bot$	$1,\bot$	$0,\bot$	$1,0,\bot$	$1,0,\bot$
$0,\bot$	$0,\bot$	$0,\bot$	$\bot$	$0,\bot$	$0,\bot$	$0,\bot$	$0,\bot$
$1,0$	$1,0$	0	$\bot$	$1,0,\bot$	$0,\bot$	$1,0$	$1,0,\bot$
$1,0,\bot$	$1,0,\bot$	$0,\bot$	$\bot$	$1,0,\bot$	$0,\bot$	$1,0,\bot$	$1,0,\bot$

Figure 4: Weak conjunction in PTaCL

$\sqcap$	P	D	NA	I(P)	I(D)	I(PD)
P	P	D	NA	I(P)	I(D)	I(PD)
D	D	D	NA	I(D)	I(D)	D,I(D)
NA	NA	NA	NA	NA	NA	NA
I(P)	I(P)	I(D)	NA	I(P)	I(D)	I(PD)
I(D)	I(D)	I(D)	NA	I(D)	I(D)	I(D)
I(PD)	I(PD)	D,I(D)	NA	I(PD)	I(D)	I(PD)

Figure 5: Weak conjunction in XACML v3. In the figure, P stands for Permit, D for Deny, NA for non-applicable, I(P) for Indeterminate{P}, I(D) for Indeterminate{D}, and I(PD) for Indeterminate{PD}.

to Figure 4. It is worth noting that two possible values are possible when combining Deny and Indeterminate{PD}. This is due to the interpretation of Indeterminate{PD}: if Indeterminate{PD} corresponds to $\{1,0\}$, the decision is Deny; otherwise, if Indeterminate{PD} corresponds to $\{1,0,\bot\}$, the decision is Indeterminate{D}. This different interpretation can lead to inaccurate or even incorrect authorization decisions.

In the next section we study the conditions under which a reduction is safe for an operator, i.e. the reduction does not change the semantics of a (composite) policy.

4. SAFE REDUCTION

As shown in the previous section, a reduction can lead to risks of inaccurate or even incorrect authorization decisions. To avoid such risks, the reduction of a decision set should be "safe". Intuitively, a reduction is safe if projecting onto decision set $\mathcal{D}_m$ the decision obtained by applying an operator *op* to two decisions in decision set $\mathcal{D}_n$ returns the same decision obtained by first projecting the two decisions onto $\mathcal{D}_m$ and then applying *op* to the resulting decisions.

In this section, we first study the safety of a reduction with respect to a combining operator. Then, we analyze the safety of a reduction with respect to a set of operators as well as the safety of a composition of reductions. The proofs of the propositions are given in Appendix.

4.1 Single operators

To understand when the reduction of a decision set is safe, we first focus on unary operators, i.e., functions $\alpha : \mathcal{D} \to \mathcal{D}$ where $\mathcal{D}$ is a decision set. For instance, the negation operator $\neg$ and the weakening operator $\sim$ over $\mathcal{D}_3$ are defined in Figure 2. Intuitively, a reduction is safe for α if the operator behaves similarly under reduction when the argument is reduced and when it is not.

Definition 2 (Safe Reduction). *Let $\mathcal{D}_n$ and $\mathcal{D}_m$ be two decision sets such that $\mathcal{D}_m \subseteq \mathcal{D}_n$, $\rho : \mathcal{D}_n \to \mathcal{D}_m$ a decision reduction, and $\alpha : \mathcal{D}_n \to \mathcal{D}_n$ a unary operator. We say that ρ is* safe *for α if and only if*

$$\forall d \in \mathcal{D}_n \quad \rho(\alpha(d)) = \rho(\alpha(\rho(d))) \tag{1}$$

Note that if $\mathcal{D}_m$ is closed under α, Eq. (1) can be simplified to $\rho(\alpha(d)) = \alpha(\rho(d))$. Figure 6 presents some unary operators together with the safety of reductions ρ^1_{32} and ρ^0_{32} defined in Section 3 with respect to these operators. We can observe that reduction ρ^1_{32} is safe neither for $\neg$ nor for $\sim$, while ρ^0_{32} is safe for $\sim$ but not for $\neg$. In practice, this means that if we have to reduce the decision set of a policy using the operator $\neg$ over sub-policies that can evaluate to $\bot$, then the result will be different whether the reduction is applied at the top-level or at the sub-policy level.

	K_{Id}	K_0	K_1	$K_\bot$	$\sqcap_0$	$\sqcup_1$	$\tilde{\sqcap}_\bot$	$\tilde{\sqcup}_\bot$	$\neg$	$\sim$
1	1	0	1	$\bot$	0	1	$\bot$	1	0	1
0	0	0	1	$\bot$	0	1	0	$\bot$	1	0
$\bot$	$\bot$	0	1	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	0
ρ^1_{32}	✓	✓	✓	✓	×	✓	✓	✓	×	×
ρ^0_{32}	✓	✓	✓	✓	✓	×	✓	✓	×	✓

Figure 6: Unary operators and safety of ρ^1_{32} and ρ^0_{32}

	$\tilde{\sqcap}_1$	$\tilde{\sqcap}_0$	$\tilde{\sqcap}_\bot$
1	1	0	$\bot$
0	0	0	0
$\bot$	$\bot$	0	$\bot$

(a) Unary operators for $\tilde{\sqcap}$

	$\sqcap_1$	$\sqcap_0$	$\sqcap_\bot$
1	1	0	$\bot$
0	0	0	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

(b) Unary operators for $\sqcap$

Figure 7: Modeling binary operators using unary operators

Using currying, any binary operator $\beta : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$ can be defined as an ordered set of unary operators $\beta = \{\alpha_1, \ldots, \alpha_n\}$, where each α_i is a unary operator. For each $d_i \in \mathcal{D}$ we write $\beta[d_i]$ for α_i such that $\beta(d_i, d_j) = \beta[d_i](d_j) = \alpha_i(d_j)$.[1] For instance, the binary strong conjunction $\tilde{\sqcap}$ can be defined as $\tilde{\sqcap} = \{\tilde{\sqcap}_1, \tilde{\sqcap}_0, \tilde{\sqcap}_\bot\}$ and the weak conjunction as $\sqcap = \{\sqcap_1, \sqcap_0, \sqcap_\bot\}$ where each $\tilde{\sqcap}_x$ and $\sqcap_x$ are defined in Figure 7a and Figure 7b, respectively.

This approach can be inductively extended to other operators: an operator $\gamma : \mathcal{D}^k \to \mathcal{D}$ with arity k can be defined as an ordered set $\{\gamma_1, \ldots, \gamma_n\}$, where each γ_i is an operator of arity $k-1$ corresponding to the decision $d_i \in \mathcal{D}$, such that $\gamma(d_1, \ldots, d_k) = \gamma[d_1](d_2, \ldots, d_k)$. The definition of safe reduction for unary operators can be easily extended to k-ary operators.

Definition 3 (Safe Reduction). *Let $\mathcal{D}_n$ and $\mathcal{D}_m$ be two decision sets such that $\mathcal{D}_m \subseteq \mathcal{D}_n$, $\rho : \mathcal{D}_n \to \mathcal{D}_m$ a decision reduction, and $\gamma : \mathcal{D}_n^k \to \mathcal{D}_n$ a k-ary operator. We say that ρ is safe for γ if and only if*

$$\forall d_1, \ldots, d_k \in \mathcal{D}_n \quad \rho(\gamma(d_1, \ldots, d_k)) = \rho(\gamma(\rho(d_1), \ldots, \rho(d_k)))$$

Figure 8 presents the safety of ρ^1_{32} and ρ^0_{32} for the conjunctive and disjunctive operators over $\mathcal{D}_3$ (defined in terms of the unary operators presented in Figure 6). We say that two operators are ρ-equivalent when they behave similarly under ρ. For instance, we can observe that the strong and weak conjunctions over $\mathcal{D}_3$ are ρ^0_{32}-equivalent, while strong

[1]For the sake of readability, we do not use the notation $\beta(d_i)(d_j)$ which, although correct, could be confusing.

	$\sqcap$	$\sqcup$	$\tilde{\sqcap}$	$\tilde{\sqcup}$
1	K_{Id}	$\sqcup_1$	K_{Id}	K_1
0	$\sqcap_0$	K_{Id}	K_0	K_{Id}
$\bot$	$K_\bot$	$K_\bot$	$\tilde{\sqcap}_\bot$	$\tilde{\sqcup}_\bot$
ρ^1_{32}	×	✓	✓	✓
ρ^0_{32}	✓	×	✓	✓

Figure 8: Binary operators and safety of ρ^1_{32} and ρ^0_{32}

and weak disjunctions over $\mathcal{D}_3$ are ρ^1_{32}-equivalent, which intuitively can be read as: if $\bot$ is treated as 0, then there is no difference between the weak and strong conjunctions, and if $\bot$ is treated as 1, then there is no difference between the weak and strong disjunctions. More formally:

DEFINITION 4. *Let γ and γ' be two operators of arity k, and $\rho : \mathcal{D}_n \to \mathcal{D}_m$ a decision reduction. We say that γ and γ' are ρ-equivalent, denoted as $\gamma \equiv_\rho \gamma'$, if and only if, for any decisions $d_1, \ldots, d_k \in \mathcal{D}_n$, we have:*

$$\rho(\gamma(d_1, \ldots, d_k)) = \rho(\gamma'(d_1, \ldots, d_k))$$

We are now in position to prove that the safety of a reduction for an operator γ of arity k can be obtained from the safety of the same reduction for the operators of arity $k-1$ that are defining γ. In practice, this result means that in order to prove the safety of a reduction for an operator, which can be quite complex for large arity, can be inductively reduced to prove the safety of the reduction for unary operators, which are simpler to analyze.

PROPOSITION 1. *Given an operator $\gamma : \mathcal{D}^k_n \to \mathcal{D}_n$ of arity k, and a decision reduction $\rho : \mathcal{D}_n \to \mathcal{D}_m$, ρ is safe for γ if and only if ρ is safe for $\gamma[d_i]$ for each $d_i \in \mathcal{D}_m$ and $\gamma[d_j] \equiv_\rho \gamma[\rho(d_j)]$ for each $d_j \in \mathcal{D}_n \setminus \mathcal{D}_m$.*

It is worth observing that it might be possible to have a safe operator γ using a non-safe operator for a decision d_j as long as both $d_j \notin \mathcal{D}_m$ and $\gamma[d_j] \equiv_\rho \gamma[\rho(d_j)]$.

Figure 8 shows that ρ^1_{32} is not safe for $\sqcap$ and ρ^0_{32} is not safe for $\sqcup$. Based on Proposition 1, the (un)safety of these reductions can be explained by analyzing their safety with respect to the unary operators forming $\sqcap$, $\sqcup$, $\tilde{\sqcap}$, and $\tilde{\sqcup}$. For instance, reduction ρ^1_{32} is safe for $\sqcap[1] = K_{Id}$, but not for $\sqcap[0] = \sqcap_0$, and therefore it is not safe for $\sqcap$ (in addition, $\sqcap[\bot] = K_\bot$ is not ρ^1_{32}-equivalent to K_{Id}). On the other hand, ρ^1_{32} is safe for $\tilde{\sqcap}[1] = K_{Id}$, safe for $\tilde{\sqcap}[0] = K_0$, and $\tilde{\sqcap}[\bot] = \tilde{\sqcap}_\bot$ is ρ^1_{32}-equivalent to K_{Id}, so it is safe for $\tilde{\sqcap}$. These examples illustrate the constructiveness of our approach: for $\sqcap$, we can directly pinpoint where the problem comes from, i.e., operator $\sqcap_0$ treats decisions 1 and $\bot$ differently, while ρ^1_{32} projects $\bot$ to 1. Thus, we can conclude ρ^1_{32} is not safe for $\sqcap$ because it is not safe for $\sqcap_0$.

4.2 Composition of operators

Typically, access control models provides more than one operator for combining decisions. We now study the safety of a reduction for a set of operators.

PROPOSITION 2. *Given operators $\gamma, \gamma_1, \ldots, \gamma_j : \mathcal{D}^k_n \to \mathcal{D}_n$ and decision reduction $\rho : \mathcal{D}_n \to \mathcal{D}_m$, if ρ is safe for $\gamma, \gamma_1, \ldots, \gamma_j$, then ρ is safe for any combination of $\gamma, \gamma_1, \ldots, \gamma_j$.*

	$\mathsf{pov_P}$	$\mathsf{pov_D}$	$\mathsf{pov_{NA}}$	$\mathsf{pov_{I(P)}}$	$\mathsf{pov_{I(D)}}$	$\mathsf{pov_{I(PD)}}$
P	P	P	P	P	P	P
D	P	D	D	I(PD)	D	I(PD)
NA	P	D	NA	I(P)	I(D)	I(PD)
I(P)	P	I(PD)	I(P)	I(P)	I(PD)	I(PD)
I(D)	P	D	I(D)	I(PD)	I(D)	I(PD)
I(PD)	P	I(PD)	I(PD)	I(PD)	I(PD)	I(PD)
ρ^{I}_{64}	✓	×	✓	✓	✓	✓

Figure 9: Unary operators for permit-overrides and safety of ρ^{I}_{64}

This result shows that, when more than one operator is used to combine decisions, it is sufficient to study the safety of the operators individually in order to determine whether a decision set can be safely reduced. For the sake of clarity, Proposition 2 is limited to the case where every operator has the same arity k. The proposition can be trivially extended to the case where every operator has an arbitrary arity.

4.3 Composition of reductions

Reductions can be composed together, in order to reduce successively a decision. For instance, consider the reduction $\rho^\bot_{43} : \mathcal{D}_4 \to \mathcal{D}_3$, which reduces $\top$ to $\bot$, i.e., $\rho^\bot_{43}(\top) = \bot$ and $\rho^\bot_{43}(d) = d$, for $d \neq \top$. We may now wish to compose this reduction with ρ^1_{32} or ρ^0_{32}, in order to obtain a reduction from $\mathcal{D}_4$ to $\mathcal{D}_2$.

PROPOSITION 3. *Given three decisions set $\mathcal{D}_l \subseteq \mathcal{D}_m \subseteq \mathcal{D}_n$, a reduction $\rho_1 : \mathcal{D}_n \to \mathcal{D}_m$ and a reduction $\rho_2 : \mathcal{D}_m \to \mathcal{D}_l$, an operator γ defined over $\mathcal{D}_n$ and closed under $\mathcal{D}_m$ (i.e., for any $\overline{d} \in \mathcal{D}^k_m$, $\gamma(\overline{d}) \in \mathcal{D}_m$), if both ρ_1 and ρ_2 are safe for γ, then their composition $\rho_2 \circ \rho_1$ is also safe for γ.*

5. ANALYSIS OF XACML V3

XACML v3 uses two decision sets, namely $\mathcal{D}_4$ and $\mathcal{D}_6$. Decisions are projected from one decision set to the other depending on the combining algorithm used to evaluate a composite policy. In Section 3, we showed that this can affect the final decision returned by the policy decision point, and thus the overall security provided by the employed access control mechanism. In this section, we analyze the combining operators provided by XACML v3 against the notion of safe reduction and provide guidelines for the selection of the minimal decision set to be used.

The reduction function used in XACML v3 from $\mathcal{D}_6$ to $\mathcal{D}_4$ projects the Indeterminate extended set to a single Indeterminate value (i.e., I). As discussed in Section 3, XACML v3 considers I to be equivalent to I(PD) (we use the abbreviations defined in Figure 5). More formally, reduction ρ^{I}_{64} is defined as $\rho^{\mathsf{I}}_{64}(\mathsf{I(P)}) = \rho^{\mathsf{I}}_{64}(\mathsf{I(D)}) = \rho^{\mathsf{I}}_{64}(\mathsf{I(PD)}) = \mathsf{I(PD)}$, and $\rho^{\mathsf{I}}_{64}(d) = d$ for $d \in \{\mathsf{P}, \mathsf{D}, \mathsf{NA}\}$.

Figures 9 and 10 present the encoding of combining algorithms permit-overrides and deny-overrides, respectively, using unary operators along with the safety of ρ^{I}_{64}. We can observe that some unary operators encoding these combining algorithms are not safe (i.e., $\mathsf{pov_D}$ for permit-overrides and $\mathsf{dov_P}$ for deny-overrides). Indeed, we have:

$$\rho^{\mathsf{I}}_{64}(\mathsf{pov_D}(\mathsf{I(D)})) = \mathsf{D} \qquad \rho^{\mathsf{I}}_{64}(\mathsf{pov_D}(\rho^{\mathsf{I}}_{64}(\mathsf{I(D)}))) = \mathsf{I(PD)}$$
$$\rho^{\mathsf{I}}_{64}(\mathsf{dov_P}(\mathsf{I(P)})) = \mathsf{P} \qquad \rho^{\mathsf{I}}_{64}(\mathsf{dov_P}(\rho^{\mathsf{I}}_{64}(\mathsf{I(P)}))) = \mathsf{I(PD)}$$

	$\mathsf{dov_P}$	$\mathsf{dov_D}$	$\mathsf{dov_{NA}}$	$\mathsf{dov_{I(P)}}$	$\mathsf{dov_{I(D)}}$	$\mathsf{dov_{I(PD)}}$
P	P	D	P	P	I(PD)	I(PD)
D	D	D	D	D	D	D
NA	P	D	NA	I(P)	I(D)	I(PD)
I(P)	P	D	I(P)	I(P)	I(PD)	I(PD)
I(D)	I(PD)	D	I(D)	I(PD)	I(D)	I(PD)
I(PD)	I(PD)	D	I(PD)	I(PD)	I(PD)	I(PD)
ρ^{I}_{64}	×	✓	✓	✓	✓	✓

Figure 10: Unary operators for deny-overrides and safety of ρ^{I}_{64}

In addition, $\mathsf{pov_{I(D)}}$ and $\mathsf{dov_{I(P)}}$ are not ρ^{I}_{64}-equivalent to $\mathsf{pov_{I(PD)}}$ and $\mathsf{dov_{I(PD)}}$, respectively. Based on the results presented in the previous section, it follows that ρ^{I}_{64} is safe neither for permit-overrides nor for deny-overrides. Thereby, XACML v3 lacks an unambiguous and well-defined semantics, which can have an impact on the final decision.

In order for XACML v3 to be founded on an unambiguous and well-defined semantics, it should use only one decision set, namely $\mathcal{D}_6$. The other combining algorithms – first-applicable (fa), deny-unless-permit (dup), permit-unless-deny (pud), and only-one-applicable (ooa) – can be easily extended to $\mathcal{D}_6$ based on their semantics on $\mathcal{D}_4$.

We now define the unary operators for each of these operators, for decision set $\{\mathsf{P}, \mathsf{D}, \mathsf{NA}, \mathsf{I(P)}, \mathsf{I(D)}, \mathsf{I(PD)}\}$. Remark that a k-ary operator can be represented as an ordered set of $k-1$-ary operators. The order of operators is defined by the order of decisions in the decision set above.

deny-unless-permit: for any $x \in \mathcal{D}_6$, let $K_{\mathsf{P}}(x) = \mathsf{P}$, and let $\mathsf{dup_D}(x) = \mathsf{P}$ if $x = \mathsf{P}$ and $\mathsf{dup_D}(x) = \mathsf{D}$ otherwise; then $\mathsf{dup} = \{K_{\mathsf{P}}, \mathsf{dup_D}, \mathsf{dup_D}, \mathsf{dup_D}, \mathsf{dup_D}, \mathsf{dup_D}\}$.

permit-unless-deny: for any $x \in \mathcal{D}_6$, let $K_{\mathsf{D}}(x) = \mathsf{D}$, and let $\mathsf{pud_P}(x) = \mathsf{D}$ if $x = \mathsf{D}$ and $\mathsf{pud_P}(x) = \mathsf{P}$ otherwise; then $\mathsf{pud} = \{\mathsf{pud_P}, K_{\mathsf{P}}, \mathsf{pud_P}, \mathsf{pud_P}, \mathsf{pud_P}, \mathsf{pud_P}\}$.

first-applicable: for any $x \in \mathcal{D}_6$ and any $y \in \mathcal{D}_6$, let $\mathsf{fa}_x(y) = x$ if $x \neq \mathsf{NA}$, and $\mathsf{fa}_x(y) = y$ otherwise; then $\mathsf{fa} = \{\mathsf{fa_P}, \mathsf{fa_D}, \mathsf{fa_{NA}}, \mathsf{fa_{I(P)}}, \mathsf{fa_{I(D)}}, \mathsf{fa_{I(PD)}}\}$.

only-one-applicable: for any $x \in \mathcal{D}_6$, let $K_{Id}(x) = x$, and for any $y \in \mathcal{D}_6$, let $\mathsf{ooa}_y(x) = y$ if $x = \mathsf{NA}$, and $\mathsf{ooa}_y(x) = \mathsf{I(PD)}$ otherwise; then $\mathsf{ooa} = \{\mathsf{ooa_P}, \mathsf{ooa_D}, K_{Id}, \mathsf{ooa_{I(P)}}, \mathsf{ooa_{I(D)}}, \mathsf{ooa_{I(PD)}}\}$.

Note that the semantics of only-one-applicable with respect to indeterminacy is undefined over $\mathcal{D}_6$; thus, different interpretations can be used to define only-one-applicable over the Indeterminate extended set. For instance, an alternative interpretation of only-one-applicable in XACML v3, hereafter denoted as $\overline{\mathsf{ooa}}$, is to record the decision of the policies that caused the indeterminacy. For instance, $\overline{\mathsf{ooa}}_{\mathsf{P}}(\mathsf{P}) = \mathsf{I(P)}$, $\overline{\mathsf{ooa}}_{\mathsf{P}}(\mathsf{D}) = \mathsf{I(PD)}$, $\overline{\mathsf{ooa}}_{\mathsf{P}}(\mathsf{I(P)}) = \mathsf{I(P)}$, etc. ($\overline{\mathsf{ooa}}$ behaves as ooa if at most one policy is applicable) In the rest of the section, we consider both interpretations.

The question is whether $\mathcal{D}_6$ is the most appropriate decision set for XACML v3. Ideally, the decision set should be the smallest as possible in order to reduce the complexity of the policy evaluation process and thus facilitate the task of making a conclusive decision. On the other hand, it should be expressive enough to capture the semantics of combining operators. Based on these observations, we say that a decision set is *minimal* for a (set of) operator(s) if there exists no safe reduction to a smaller decision set.

fa	1	0	⊥
1	1	1	1
0	0	0	0
⊥	1	0	⊥

(a) first-applicable over $\mathcal{D}_3$

$\overline{\mathsf{fa}}$	1	0	⊥	1,⊥	0,⊥	1,0	1,0,⊥
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
⊥	1	0	⊥	1,⊥	0,⊥	1,0	1,0,⊥
1,⊥	1	1,0	1,⊥	1,⊥	1,0,⊥	1,0	1,0,⊥
0,⊥	1,0	0	0,⊥	1,0,⊥	0,⊥	1,0	1,0,⊥
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
1,0,⊥	1,0	1,0	1,0,⊥	1,0,⊥	1,0,⊥	1,0	1,0,⊥

(b) Operator $\overline{\mathsf{fa}}$ over $\mathcal{D}_7$

Figure 11: Defining an operator resembling first-applicable over $\mathcal{D}_7$

	pov	dov	fa	dup	pud	ooa	$\overline{\mathsf{ooa}}$	$\overline{\mathsf{fa}}$	⊓	⊔	$\tilde{\sqcap}$	$\tilde{\sqcup}$
ρ^{I}_{76}	✓	✓	✓	✓	✓	✓	✓	✓	×	×	✓	✓
ρ^{I}_{64}	×	×	✓	✓	✓	✓	✓	×	×	×	×	×

Figure 12: Safety of reduction functions with respect to operators over $\mathcal{D}_6$ and $\mathcal{D}_7$

We use an operator $\overline{\mathsf{fa}}$ over $\mathcal{D}_7$ resembling first-applicable in XACML to discuss the minimality of the decision set for XACML v3. To define this operator, we can first define first-applicable in $\mathcal{D}_3$ (Figure 11a) and then extend it to $\mathcal{D}_7$ in a point-wise way as described in Section 2.3.1. The decision table for $\overline{\mathsf{fa}}$ over $\mathcal{D}_7$ is presented in Figure 11b. We can observe that $\overline{\mathsf{fa}}$ returns a conclusive decision in cases where fa returns an Indeterminate decision. In particular, $\overline{\mathsf{fa}}(\{1,\bot\},\{1\}) = 1$ while $\mathsf{fa}(\mathsf{I(P)},\mathsf{P}) = \mathsf{I(P)}$; also $\overline{\mathsf{fa}}(\{0,\bot\},\{0\}) = 0$ while $\mathsf{fa}(\mathsf{I(D)},\mathsf{D}) = \mathsf{I(D)}$. Intuitively, this means that, if the first applicable policy is evaluated to Permit (Deny resp.) in case an error would not have occurred and the next applicable policy is evaluated to Permit (Deny resp.), then we have sufficient evidence to grant permission. Notice that this intuition is already adopted in XACML v3 for the target evaluation. In particular, if at least one element ⟨AllOf⟩ matches the query, then the element ⟨AnyOf⟩ containing that ⟨AllOf⟩ evaluates to 'Match' regardless if an error occurs in another element ⟨AllOf⟩ of the same ⟨AnyOf⟩ [2, p. 81-82]. We argue that the semantics of $\overline{\mathsf{fa}}$ is preferable to the one of fa as it provides a conclusive decision in more cases without changing the intended semantics of the composite policy. Let now consider the reduction from $\mathcal{D}_7$ to $\mathcal{D}_6$ defined in Section 3, hereafter denoted $\rho^{\mathsf{I(PD)}}_{76}$. We can easily verify that $\rho^{\mathsf{I(PD)}}_{76}$ is safe with respect to $\overline{\mathsf{fa}}$. This might confirm that $\mathcal{D}_6$ is the appropriate decision set for XACML v3 also if operator $\overline{\mathsf{fa}}$ is used instead of fa.

In general, the choice of the decision set to be used depends on the employed operators. Figure 12 shows the safety of reduction functions $\rho^{\mathsf{I(PD)}}_{76}$ and ρ^{I}_{64} for a number of operators. Operators fa, dup, pud, ooa and $\overline{\mathsf{ooa}}$ can be defined over $\mathcal{D}_7$ as we did for $\mathcal{D}_6$. Operators pov and dov can be defined over $\mathcal{D}_7$ in a point-wise way. It is worth noting that

a point-wise definition of pov and dov over $\mathcal{D}_7$ behaves as pov and dov as defined in XACML.

Based on Figure 12, we can conclude that $\mathcal{D}_6$ is the minimal decision set with respect to the operators used in XACML v3. Indeed, $\mathcal{D}_7$ can be safely reduced to $\mathcal{D}_6$ with respect to these operators. However, if we want to extend XACML with additional operators already defined for a larger decision set, it is necessary to determine the safety of the reduction function with respect to the new operators. For instance, suppose we want to support weak conjunction ($\sqcap$) and weak disjunction ($\sqcup$) in XACML. Reduction $\rho_{76}^{\mathsf{I(PD)}}$ is not safe with respect to these operators. Therefore, the minimal decision set that preserves the semantics of a policy would be $\mathcal{D}_7$. Conversely, $\mathcal{D}_4$ is sufficient to support an access control model with operators fa, dup, pud, and ooa (or $\overline{\mathsf{ooa}}$).

The different reductions presented up to now can be chained together; for instance, we can reduce from $\mathcal{D}_7$ to $\mathcal{D}_6$ with $\rho_{76}^{\mathsf{I(PD)}}$; from $\mathcal{D}_6$ to $\mathcal{D}_4$ with ρ_{64}^{I}; from $\mathcal{D}_4$ to $\mathcal{D}_3$ with $\rho_{43}^{\perp}$; and from $\mathcal{D}_3$ to $\mathcal{D}_2$ with ρ_{32}^{0}. Hence, we can observe that since all reductions are safe for the operators pud and dup, then the composite reduction $\rho_{72}^{0} = \rho_{32}^{0} \circ \rho_{43}^{\perp} \circ \rho_{64}^{\mathsf{I}} \circ \rho_{76}^{\mathsf{I(PD)}}$ is also safe for these operators, following Proposition 3. Note that this proposition is not an equivalence, and we can observe that ρ_{72}^{0} is safe for $\sqcap$ even though ρ_{64}^{I} and $\rho_{76}^{\mathsf{I(PD)}}$ are not.

6. CONCLUSIONS

Although reducing a decision set may be needed, for instance to obtain a conclusive decision, it may change the semantics of a composite policy, leading to accept a query that would have been intuitively denied (or vice versa). This paper has presented a formal framework centered on the notion of decision reduction for determining whether a decision reduction preserves the intended semantics of a policy. We demonstrated the framework by analyzing XACML v3, the de facto standard for the specification and enforcement of access control policies. XACML v3 defines combining operators over two decisions sets, namely $\mathcal{D}_4$ and $\mathcal{D}_6$. Depending on the operator used to evaluate a policy, a decision set is projected onto the other set. Our analysis shows that the reduction function used in XACML v3, which projects the Indeterminate extended set to a single Indeterminate value, may result in an unnecessary loss of accuracy, which affects the final access decision. In particular, this reduction is not safe with respect to permit-overrides and deny-overrides. We show that the choice of the decision set depends on the combining operators used in the access control model. For instance, extending an access control model by reusing an operator defined over a larger decision set requires ensuring that the reduction does not change the semantics of the operator, i.e. the reduction is safe for that operator.

Acknowledgement This work has been partially funded by the EDA project IN4STARS2.0, the ITEA2 project FedSS, and the ARTEMIS project ACCUS, the NWO CyberSecurity programme under the PriCE project, the Dutch national program COMMIT under the THeCS project and the Choice Architecture for Information Security project, EPSRC Grant EP/K006568/1.

7. REFERENCES

[1] eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard, OASIS, 2005.

[2] eXtensible Access Control Markup Language (XACML) Version 3.0. OASIS Standard, OASIS, 2012.

[3] O. Arieli and A. Avron. The value of the four values. *Artificial Intelligence*, 102(1):97–141, 1998.

[4] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise Privacy Authorization Language (EPAL). Technical report, IBM Research, Rüschlikon, 2003.

[5] D. E. Bell and L. J. LaPadula. Secure computer systems: A mathematical model, Volume II. *Journal of Computer Security*, 4(2/3):229–263, 1996.

[6] D. F. C. Brewer and M. J. Nash. The Chinese Wall Security Policy. In *Proceedings of Symposium on Security and Privacy*, pages 329–339. IEEE, 1989.

[7] G. Bruns and M. Huth. Access control via Belnap logic: Intuitive, expressive, and analyzable policy composition. *TISSEC*, 14(1):9, 2011.

[8] P.-C. Cheng, P. Rohatgi, C. Keser, P. A. Karger, G. M. Wagner, and A. S. Reninger. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *Proceedings of Symposium on Security and Privacy*, pages 222–230. IEEE, 2007.

[9] J. Crampton and M. Huth. An authorization framework resilient to policy evaluation failures. In *Computer Security*, LNCS 6345, pages 472–487. Springer, 2010.

[10] J. Crampton and C. Morisset. PTaCL: A language for attribute-based access control in open systems. In *Proceedings of POST*, LNCS 7215, pages 390–409. Springer, 2012.

[11] S. De Capitani Di Vimercati, S. Foresti, P. Samarati, and S. Jajodia. Access control policies and languages. *Int. J. Comput. Sci. Eng.*, 3(2):94–102, 2007.

[12] D. F. Ferraiolo and D. R. Kuhn. Role-based access control. In *Proceedings of the 15th National Computer Security Conference*, pages 554–563, 1992.

[13] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, 1976.

[14] W. Jobe. Functional completeness and canonical forms in many-valued logics. *Journal of Symbolic Logic*, 27(4):409–422, 1962.

[15] B. W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8(1):18–24, 1974.

[16] N. Li, Q. Wang, W. H. Qardaji, E. Bertino, P. Rao, J. Lobo, and D. Lin. Access control policy combining: theory meets practice. In *Proceedings of 14th ACM SACMAT*, pages 135–144. ACM, 2009.

[17] Q. Ni, E. Bertino, and J. Lobo. D-algebra for composing access control policy decisions. In *Proceedings of ACM AsiaCCS*, pages 298–309. ACM, 2009.

[18] P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo. An algebra for fine-grained integration of XACML policies. In *Proceedings of 14th ACM SACMAT*, pages 63–72. ACM, 2009.

[19] D. Trivellato, N. Zannone, M. Glaundrup, J. Skowronek, and S. Etalle. A semantic security

framework for systems of systems. *Int. J. Cooperative Inf. Syst.*, 22(1), 2013.

[20] M. C. Tschantz and S. Krishnamurthi. Towards reasonability properties for access-control policy languages. In *Proceedings of 11th ACM SACMAT*, pages 160–169. ACM, 2006.

[21] T. Y. C. Woo and S. S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security*, 2(2-3):107–136, 1993.

APPENDIX

A. PROOFS

In this section we provide the proofs of the propositions in Section 4.

Proof (Proposition 1). Let us first observe that for any decision $d_i \in \mathcal{D}_m$, since $d_i = \rho(d_i)$ by definition of ρ, we trivially have $\gamma[d_i] \equiv_\rho \gamma[\rho(d_i)]$.

($\Leftarrow$) Given $d_1, \dots d_k \in \mathcal{D}_n$,:

$$\begin{aligned}\rho(\gamma(d_1,\dots,d_k)) &= \rho(\gamma[d_1](d_2,\dots,d_k)) \\ &\quad \text{(definition of } \gamma) \\ &= \rho(\gamma[\rho(d_1)](d_2,\dots,d_k)) \\ &\quad (\gamma[\rho(d_1)] \equiv_\rho \gamma[d_1]) \\ &= \rho(\gamma[\rho(d_1)](\rho(d_2),\dots,\rho(d_k))) \\ &\quad (\rho(d_1) \in \mathcal{D}_m, \text{ so } \rho \text{ is safe for } \gamma[\rho(d_1)]) \\ &= \rho(\gamma(\rho(d_1),\rho(d_2),\dots,\rho(d_k))) \\ &\quad \text{(definition of } \gamma)\end{aligned}$$

($\Rightarrow$) Let γ be a safe operator. Let us first prove that, for any $d_i \in \mathcal{D}_m$, $\gamma[d_i]$ is safe. Given $d_1, \dots, d_{k-1} \in \mathcal{D}_n$, we have:

$$\begin{aligned}\rho(\gamma[d_i](d_1,\dots,d_{k-1})) &= \rho(\gamma(d_i,d_1,\dots,d_{k-1})) \\ &\quad \text{(definition of } \gamma) \\ &= \rho(\gamma(\rho(d_i),\rho(d_1),\dots,\rho(d_{k-1}))) \\ &\quad (\rho \text{ is safe for } \gamma) \\ &= \rho(\gamma[\rho(d_i)](\rho(d_1),\dots,\rho(d_{k-1}))) \\ &\quad \text{(definition of } \gamma) \\ &= \rho(\gamma[d_i](\rho(d_1),\dots,\rho(d_{k-1}))) \\ &\quad (d_i = \rho(d_i) \text{ by definition of } \rho)\end{aligned}$$

Now, given $d_j \in \mathcal{D}_n \setminus \mathsf{D}_m$, let us prove that $\gamma[d_j] \equiv_\rho \gamma[\rho(d_j)]$. Let $d_1, \dots, d_{k-1} \in \mathcal{D}_n$:

$$\begin{aligned}\rho(\gamma[d_j](d_1,\dots,d_{k-1})) &= \rho(\gamma(d_j,d_1,\dots,d_{k-1})) \\ &\quad \text{(definition of } \gamma) \\ &= \rho(\gamma(\rho(d_j),\rho(d_1),\dots,\rho(d_{k-1}))) \\ &\quad (\rho \text{ is safe for } \gamma) \\ &= \rho(\gamma[\rho(d_j)](\rho(d_1),\dots,\rho(d_{k-1}))) \\ &\quad \text{(definition of } \gamma) \\ &= \rho(\gamma[\rho(d_j)](d_1,\dots,d_{k-1})) \\ &\quad (\rho(d_j) \in \mathcal{D}_m, \text{ so } \rho \text{ safe for } \gamma[\rho(d_j)])\end{aligned}$$

□

Proof (Proposition 2). Let us denote $\overline{d}$ the arguments of an operator γ_i, i.e. $\gamma_i(\overline{d})$ stands for $\gamma_i(d_1,\dots,d_k)$. Also, we write $\rho(\overline{d})$ for $\rho(d_1),\dots\rho(d_k)$. We have:

$$\begin{aligned}\rho(\gamma(\gamma_1(\overline{d}),\dots,\gamma_k(\overline{d}))) &= \rho(\gamma(\rho(\gamma_1(\overline{d})),\dots,\rho(\gamma_k(\overline{d})))) \\ &\quad (\rho \text{ safe for } \gamma) \\ &= \rho(\gamma(\rho(\gamma_1(\rho(\overline{d})),\dots,\rho(\gamma_k(\rho(\overline{d})))))) \\ &\quad (\rho \text{ safe for } \gamma_1,\dots,\gamma_k) \\ &= \rho(\gamma(\gamma_1(\rho(\overline{d})),\dots,\gamma_k(\rho(\overline{d})))) \\ &\quad (\rho \text{ safe for } \gamma)\end{aligned}$$

□

Proof Proposition 3. Let $\overline{d} \in \mathcal{D}_n^k$:

$$\begin{aligned}\rho_2(\rho_1(\gamma(\overline{d}))) &= \rho_2(\rho_1(\gamma(\rho_1(\overline{d})))) && (\rho_1 \text{ safe for } \gamma) \\ &= \rho_2(\gamma(\rho_1(\overline{d}))) && (\mathcal{D}_m \text{ closed under } \rho_1) \\ &= \rho_2(\gamma(\rho_2(\rho_1(\overline{d})))) && (\rho_2 \text{ safe for } \gamma) \\ &= \rho_2(\rho_1(\gamma(\rho_2(\rho_1(\overline{d}))))) && (\mathcal{D}_m \text{ closed under } \rho_1)\end{aligned}$$

□