PTaCL: A Language for Attribute-Based Access Control in Open Systems

Jason Crampton[1] and Charles Morisset[1,2,⋆]

[1] Information Security Group,
Royal Holloway, University of London,
Egham, Surrey TW20 0EX, U.K.
Jason.Crampton@rhul.ac.uk
[2] Security Group,
Istituto di Informatica e Telematica (IIT), C.N.R.,
Via Giuseppe Moruzzi, 1, 56124 Pisa, Italy
Charles.Morisset@iit.cnr.it

Abstract. Many languages and algebras have been proposed in recent years for the specification of authorization policies. For some proposals, such as XACML, the main motivation is to address real-world requirements, typically by providing a complex policy language with somewhat informal evaluation methods; others try to provide a greater degree of formality – particularly with respect to policy evaluation – but support far fewer features. In short, there are very few proposals that combine a rich set of language features with a well-defined semantics, and even fewer that do this for authorization policies for attribute-based access control in open environments. In this paper, we decompose the problem of policy specification into two distinct sub-languages: the policy target language (PTL) for target specification, which determines when a policy should be evaluated; and the policy composition language (PCL) for building more complex policies from existing ones. We define syntax and semantics for two such languages and demonstrate that they can be both simple and expressive. PTaCL, the language obtained by combining the features of these two sub-languages, supports the specification of a wide range of policies. However, the power of PTaCL means that it is possible to define policies that could produce unexpected results. We provide an analysis of how PTL should be restricted and how policies written in PCL should be evaluated to minimize the likelihood of undesirable results.

Keywords: Target, Policy, Composition, PCL, PTL, PTaCL.

1 Introduction

One of the fundamental security services in computer systems is *access control*, a mechanism for constraining the interaction between (authenticated) users and

⋆ Work partially supported by EU FP7-ICT project NESSoS (Network of Excellence on Engineering Secure Future Internet Software Services and Systems) under the grant agreement n. 256980.

P. Degano and J.D. Guttman (Eds.): POST 2012, LNCS 7215, pp. 390–409, 2012.
© Springer-Verlag Berlin Heidelberg 2012

protected resources. Generally, access control is implemented by an authorization service, which includes an *authorization decision function* (ADF) for deciding whether a user request to access a resource (an "access request") should be permitted or not. In its simplest form an authorization decision function either returns an allow or a deny decision.

Many access control models and systems are *policy-based*, in the sense that a request for access to protected resources is evaluated with respect to a policy that defines which requests are authorized. Many languages have been proposed for the specification of authorization policies, perhaps the best known being XACML [3,8,14]. However, it is generally acknowledged that XACML suffers from having poorly defined and counterintuitive semantics [12,13]. More formal approaches have provided well-defined semantics and typically use "policy operators" to construct complex policies from simpler sub-policies [1,4,17]. However, such approaches tend to support fewer "features" than XACML.

In a "closed" information system – one in which all authorized users are known to the system – it is possible to authenticate users of the system and to ascribe an identity to processes associated with those users. Hence, access control decisions and the policies that inform those decisions can be based on user identifiers.

Increasingly, it is necessary to define authorization policies for "open" systems, where we must make access control decisions based on user attributes, rather than identities. Hence, access request formats need to change from the user-centric subject-object-action triples of classical access control models [2,9], although such request formats are still widely used in the specification of access control models and authorization policy languages [3,4,8,14,17].

An authorization policy is typically defined by a target, a set of child policies and a decision-combining algorithm. The target, either implicitly or explicitly, identifies a set of requests. The policy is said to be "applicable" if the access request belongs to (or "matches") the target. If a policy is applicable, then its child policies are evaluated and the results returned by those child policies are combined using the decision-combining algorithm.

Informally, a policy may be regarded as a tree, in which the leaf nodes return a "conclusive" decision (allow or deny). If a request does not match the target of a leaf policy then the evaluation of that policy returns a "not applicable" decision. Hence, the set of possible decisions is 3-valued.

However, it may be the case that it is not possible to evaluate request applicability: perhaps the simplest case arises when the request is malformed. But once the request format is extended to accommodate attribute-based access control, the problem of evaluating the applicability of a request becomes even more acute. In other words, the result of request applicability is not necessarily binary: in particular, we must include a value that represents that some error has occurred while trying to evaluate request applicability. Naturally, extending the set of results that can be returned when evaluating request applicability means that we need to reconsider policy evaluation.

We believe that existing proposals for authorization policy languages suffer from at least one of the following problems:

- no support for attribute-based requests (and hence attribute-based authorization policies);
- a lack of formality in the definition of target and policy evaluation, leading to ambiguity about the meaning of policies;
- a poor understanding of the way in which attribute-based requests, targets and policies interact.

Our main objective is to define a policy language that addresses the same problem space as XACML 3.0 [15] while retaining the formality of recent work on policy algebras [1,4,5,6,17]. XACML (eXtensible access control markup language) is a standardized language: XACML 2.0 was ratified in 2005; XACML 3.0 will add support for attribute-based access control and policy administration. More specifically, our objectives are:

- to define a request format that is appropriate for attribute-based authorization policies;
- to define a syntax for specifying policy targets;
- to formally define an evaluation method for those targets that is sufficiently robust to withstand deliberate attempts to exploit the greater freedom provided by our request format;
- to define a syntax for policies, which makes use of the policy target language;
- to formally define an evaluation method for those policies that is able to handle errors in target evaluation gracefully and securely.

In this paper, we develop two distinct languages for completely defining authorization policies. Roughly speaking, our goals are to combine support for the wide variety of policies that can be defined in more informal approaches such as XACML with the more formal semantics with which policy algebras are furnished. Our policy target language (PTL) provides a syntax for specifying policy targets, while our policy composition language (PCL), provides a language for combining policies (that is, constructing policy trees). Together, we call this PTaCL, read "p-tackle", to denote *policy target and composition language*. We also provide "authorization policy semantics", which enable us to ascribe a meaning to a policy for a given request. That meaning is determined by the target semantics and the composition semantics.

The main contribution of this work is therefore the definition of PTaCL, which, although far simpler syntactically than XACML 2.0 and 3.0, can express any desired target or policy, thanks to the functional completeness of PTL and PCL. We specify precisely how to evaluate any target and policy expressed in PTaCL, thus providing the basis for a low-level language into which XACML policies, for example, could be compiled and evaluated. Moreover, we identify the problem of *attribute-hiding attacks*, where a user deliberately suppresses attributes in order to gain favorable authorization decisions, and we propose different restrictions on the definition of a target in order to avoid such attacks. We note that such attacks are not peculiar to PTaCL; they are a potential problem for any attribute-based

access control mechanism. We believe we are the first to identify and, therefore, propose mitigation strategies for, this type of attack.

In the next section, we define our request format and illustrate some of the challenges introduced by attribute-based access control. Then, in Section 3, we define the syntax and evaluation method for targets. In Section 4, we define policy syntax and evaluation. In this section, we reflect on the problems that might arise because of the more flexible request format we use and explain how those problems inform the development of PTaCL. We also explain how PTL can be restricted to provide certain guarantees about the decisions returned by policy evaluation, thereby addressing the problem of attribute-hiding attacks. We conclude the paper with a discussion of related work and some ideas for future work.

2 Attribute-Based Requests

The simplest authorization policy languages assume that an access request comprises three identifiers: the requester, the resource to which access is requested, and the type of the requested interaction (such as read, write, etc), often known as subject, object and action, respectively. The authorization decision function (ADF) associated with a given language will take that request and an authorization policy as input and return a decision. For more complex languages, the ADF may require additional information, such as the roles or security groups associated with a user, in order to make a decision. These attributes may be "pushed" with the request or "pulled" from authoritative information sources (such as the policy information points in the XACML architecture). The increasingly "open" nature of distributed computer systems, where the user population is not known in advance, requires authorization languages that are not based on user identities. For this reason, attribute-based access control (ABAC) and languages that support ABAC are expected to become increasingly important.

PTaCL comprises two sub-languages: PTL for target specification and PCL for policy specification. Policies written in PTaCL are used to evaluate access requests that may contain arbitrary attributes associated with users, resources and actions.

We model a request as a set of name-value pairs, where each name specifies an attribute and each value specifies a value for the corresponding attribute. In the simplest situation, for example, we might have attribute names such as `subject`, `object` and `action`, and a request might have the form

$$\{(\texttt{subject}, \mathit{alice}), (\texttt{object}, \mathit{test.txt}), (\texttt{action}, \mathit{read})\}\,.$$

The above request is no different from the usual view of an access request as a subject-object-action triple. However, the request format described above is not limited to requests of this form and can be used to represent requests that do not contain identifiers for subjects, objects and actions. We could, for example, have a request of the form

$$\{(\texttt{role}, \mathit{nurse}), (\texttt{object}, \mathit{test.txt}), (\texttt{action}, \mathit{read})\}\,.$$

An attribute name may appear multiple times in the request; the above request could include multiple role identifiers, for example. The use of some set of name-value pairs, rather than the fixed format subject-object-action triples (as used in XACML 2.0 [14] and most other policy languages), means that we can specify targets and requests with greater freedom than is usually the case. However, the greater freedom with which requests can be specified also means that we have to take greater care in the specification of policies.

As an example, we consider a simplified instance of the Chinese Wall policy, where a company A defines a policy to protect a set of confidential resources. Informally, this policy states that if a user is working for A, then she can access the (confidential) resource o, unless she is also working for B, the direct competitor of A, in which case the access is denied. We consider the following requests:

$$r_1 = \{(\texttt{employer}, A), (\texttt{confidential}, true)\}\,;$$
$$r_2 = \{(\texttt{employer}, A), (\texttt{employer}, B), (\texttt{confidential}, true)\}\,;$$
$$r_3 = \{(\texttt{confidential}, false)\}\,;$$
$$r_4 = \{(\texttt{confidential}, true)\}\,.$$

Informally, an ABAC policy defines a set of atomic policies (or rules), where each atomic policy describes the subset of requests to which it applies – the policy's *target* – and the decision to take when it is applicable. When a request does not belong to the policy's target, then this policy is non-applicable, which has a different meaning from saying that the request is denied. The decisions returned by the evaluation of the atomic policies are then combined together using decision combination operators.

For instance, the policy enforced by the company A should comprise two rules, the second of which is applicable to all requests and returns allow. The first rule is applicable if the request contains $(\texttt{confidential}, true)$, and in this case, if the user works for A, then it is allowed, unless she also works for B, in which case it is denied. The two rules are combined using a deny-overrides combination operator. The first rule would not be applicable to request r_3 and hence the request would be allowed. The first rule would be applicable to the remaining requests. Therefore, the evaluation of r_1 would return allow, while the evaluation of request r_2 would return deny.

Note that if the user is able to *suppress* the element $(\texttt{employer}, B)$ in r_2, then the resulting request would be allowed. We call such a situation a *partial attribute hiding attack*, where, by hiding some of her attributes, a user is able to obtain a more favorable authorization decision. A second possibility is for the user to suppress all the employer attributes. Hence, we might wish to insist that if the resource is confidential, then the request *must* contain information about the employer(s) of the requesting user, otherwise the evaluation of the request should fail. In particular, r_4 must not be allowed, returning either deny or some appropriate evaluation-error decision.

We now describe PTaCL, which provides mechanisms to tackle the issues raised by this simple example, in particular by considering attribute requests

instead of subject-object-action requests; by distinguishing between optional and mandatory attributes; and by stating two properties of monotonicity, thus allowing the detection of policies vulnerable to partial attribute hiding attacks.

3 Targets

We first define a syntax for targets. Then, in Section 3.1, we will define how to evaluate a target with respect to a request. We define three types of *atomic target*:

- null_T is a target;
- n is a target, where n is an attribute name;
- (n, v, f) is a target, where n is an attribute name, v is an attribute value and f is a binary predicate.

The most usual predicate is likely to be a test for (string) equality, but other predicates, such as $\leqslant$, $<$, $\geqslant$ and $>$, are possible. For ease of exposition, we assume throughout that all attributes are of type string and that f is string equality; henceforth we omit f from the definition of an atomic target.

We build more complex targets by defining two binary target operators, and_T and or_T, and two unary target operators, opt_T and not_T. Let t, t_1 and t_2 be targets. Then the following terms are also targets:

$$\mathsf{opt}_\mathrm{T}\, t, \quad \mathsf{not}_\mathrm{T}\, t, \quad (t_1 \,\mathsf{and}_\mathrm{T}\, t_2) \quad \text{and} \quad (t_1 \,\mathsf{or}_\mathrm{T}\, t_2).$$

The operators opt_T and not_T bind more tightly than and_T and or_T: $\mathsf{opt}_\mathrm{T}\, t \,\mathsf{and}_\mathrm{T}\, t'$, for example, is interpreted as $(\mathsf{opt}_\mathrm{T}\, t) \,\mathsf{and}_\mathrm{T}\, t'$, rather than $\mathsf{opt}_\mathrm{T}(t \,\mathsf{and}_\mathrm{T}\, t')$. As we will see in Section 3.1, the semantics of or_T and and_T are provided by associative, commutative binary operators on the set of target evaluation decisions, so we can (and will) omit brackets from expressions of the form $(t_1 \,\mathsf{or}_\mathrm{T}\, (t_2 \,\mathsf{or}_\mathrm{T} \ldots \mathsf{or}_\mathrm{T}\, t_k))$ and $(t_1 \,\mathsf{and}_\mathrm{T}\, (t_2 \,\mathsf{and}_\mathrm{T} \ldots \mathsf{and}_\mathrm{T}\, t_k))$.

In Section 4, we will define similar operators for policies and use a subscript P to distinguish them from target operators. When no ambiguity can occur we will omit the subscripts T and P.

3.1 Evaluation

A target is evaluated with respect to a request, represented as a set of name-value pairs (as described in Section 2). Informally, a request is said to "match" an atomic target if the name of one of the attribute pairs in the request is the same as the name defined in the atomic target and the predicate f evaluated at v and the corresponding value in the request is true. If no such pair exists in the request, then the request does not match the target.

The "universal" target null is matched by all requests; the target n is matched by all requests that include an attribute pair (n, v) for any value v; the target (n, v) is matched by any request that includes the specific attribute pair (n, v). The target `employer`, for example, is matched by requests r_1 and r_2 defined in Section 2 but not by the requests r_3 and r_4.

$\sqcap$	1_T	0_T	$\bot_T$
1_T	1_T	0_T	$\bot_T$
0_T	0_T	0_T	$\bot_T$
$\bot_T$	$\bot_T$	$\bot_T$	$\bot_T$

(a)

$\sqcup$	1_T	0_T	$\bot_T$
1_T	1_T	1_T	$\bot_T$
0_T	1_T	0_T	$\bot_T$
$\bot_T$	$\bot_T$	$\bot_T$	$\bot_T$

(b)

$\tilde{\sqcap}$	1_T	0_T	$\bot_T$
1_T	1_T	0_T	$\bot_T$
0_T	0_T	0_T	0_T
$\bot_T$	$\bot_T$	0_T	$\bot_T$

(c)

$\tilde{\sqcup}$	1_T	0_T	$\bot_T$
1_T	1_T	1_T	1_T
0_T	1_T	0_T	$\bot_T$
$\bot_T$	1_T	$\bot_T$	$\bot_T$

(d)

X	$\neg X$	$\sim X$
1_T	0_T	1_T
0_T	1_T	0_T
$\bot_T$	$\bot_T$	0_T

(e)

Fig. 1. Binary and unary operators on the target decision set $\{1_T, 0_T, \bot_T\}$

In addition, we may wish to distinguish the case where the request does not include the attribute name at all from the case where the attribute name was found, but with a value that does not match. Consider the atomic target $(\texttt{employer}, B)$: then request r_1 has a matching attribute name (`employer`), but $A \neq B$; in contrast, requests r_3 and r_4 do not include any matching attribute.

Informally, a request must match both t_1 and t_2 for it to match target (t_1 **and** t_2), while a request is only required to match one of t_1 and t_2 for it to match target (t_1 **or** t_2). By default, a request is required to match a target t; we can relax this requirement, while retaining the possibility of matching t, by writing **opt** t.

More formally, we define the set of target evaluation decisions Dec_T to be $\{1_T, 0_T, \bot_T\}$, where $\bot_T$ denotes that a request does not include the attribute name, 1_T denotes that a request matches an atomic target, and 0_T denotes that a request includes the attribute name but the predicate doesn't hold.[1]

We define the binary operators $\sqcap$, $\sqcup$, $\tilde{\sqcap}$ and $\tilde{\sqcup}$ on $\{1_T, 0_T, \bot_T\}$ in Fig. 1. These operators correspond to the weak and strong Kleene operators [11], respectively. We also define two unary operators $\neg$ and $\sim$ in Fig. 1. Finally, we define the total order $1_T > 0_T > \bot_T$ on Dec_T and let $\dot{\sqcup}$ denote the least upper bound operator on this ordered set.

Given a request q, we write $[\![t]\!]_T(q)$ to denote the evaluation of t with respect to q. That is, $[\![t]\!]_T(q) \in \mathsf{Dec}_T$. As for target operators, we will omit the subscript T where no ambiguity can arise. First, we define, for all requests q and for all attributes n and all values v,

$$[\![\mathsf{null}]\!](q) = 1_T \qquad \text{and} \qquad [\![n]\!](\emptyset) = [\![(n, v)]\!](\emptyset) = \bot_T.$$

We then define the evaluation of targets n and (n, v) recursively.

$$[\![n]\!](\{(n', v')\} \cup q) = \begin{cases} 1_T & \text{if } n = n' \\ [\![n]\!](q \setminus \{(n', v')\}) & \text{otherwise.} \end{cases}$$

[1] We will use analogous notation for policy-evaluation decisions, where 1_P will denote an "allow" decision and 0_P will denote a "deny" decision. The symbol $\bot$ will be used to denote an evaluation error condition in the context of targets and "not-applicable" in the context of policies.

$$[\![(n,v)]\!](\{(n',v')\} \cup q) = \begin{cases} 1_{\mathrm{T}} & \text{if } n = n',\ v = v' \\ 0_{\mathrm{T}} \dot{\sqcup} [\![(n,v)]\!](q) & \text{if } n = n',\ v \neq v' \\ [\![(n,v)]\!](q \setminus \{(n',v')\}) & \text{otherwise.} \end{cases}$$

Note that, for all q, $[\![n]\!](q)$ is either 1_{T} or $\bot_{\mathrm{T}}$. In evaluating (n,v), we compare each element of the request with the atomic target and do one of the following: we return 1_{T} if a match is found; if the attribute name matches but the predicate doesn't hold then we record the fact that the attribute name matched and continue processing; otherwise, we simply continue processing.

Since $\dot{\sqcup}$ is a supremum operator, it is commutative and associative and hence can be applied to any non-empty subset of $\mathsf{Dec}_{\mathrm{T}}$ without ambiguity. Hence, for a non-empty request $q = \{(n_1,v_1),\ldots(n_k,v_k)\}$, it is easy to see that we have

$$[\![n]\!](q) = \dot{\sqcup}\,\{[\![n]\!](\{(n_i,v_i)\}) : 1 \leqslant i \leqslant k\}\,;$$
$$[\![(n,v)]\!](q) = \dot{\sqcup}\,\{[\![(n,v)]\!](\{(n_i,v_i)\}) : 1 \leqslant i \leqslant k\}\,.$$

In other words, we can evaluate the applicability of a request with respect to a target by splitting the request into single name-value pairs and evaluating each of these requests separately. This, in turn, suggests that the evaluation of requests can be parallelized, with different target evaluation functions (TEFs) specialized for the evaluation of requests for particular attribute names.

We then define the semantics of $\mathsf{not}\,t$, $\mathsf{opt}\,t$, t_1 and t_2 and t_2 or t_2 as follows:

$$[\![\mathsf{not}\,t]\!](q) = \neg[\![t]\!](q) \qquad [\![t_1 \ \mathsf{and}\ t_2]\!](q) = [\![t_1]\!](q) \sqcap [\![t_2]\!](q)$$
$$[\![\mathsf{opt}\,t]\!](q) = \sim[\![t]\!](q) \qquad [\![t_1 \ \mathsf{or}\ t_2]\!](q) = [\![t_1]\!](q) \,\tilde{\sqcup}\, [\![t_2]\!](q)$$

Here we see that opt modifies the target t by converting a $\bot_{\mathrm{T}}$ decision (missing attribute) into a 0_{T} decision (attribute not matched). The target `opt role`, for example, evaluates to 1_{T} if a request contains a role attribute pair and evaluates to 0_{T} (rather than $\bot_{\mathrm{T}}$) if no such pair is present in the request.

It is important to note that the semantics for the and operator are provided by weak conjunction $\sqcap$, not by $\tilde{\sqcap}$. The point here is that a target is specified as part of a policy and it should not be possible to force target evaluation to return 0_{T} when the target is a conjunction and at least one of the conjuncts is mandatory. (Had we combined targets using $\tilde{\sqcap}$, if t_1 were to evaluate to 0_{T} and t_2 were to evaluate to $\bot_{\mathrm{T}}$, then $t_1 \,\tilde{\sqcap}\, t_2$ would evaluate to 0_{T}, not the desired $\bot_{\mathrm{T}}$.)

3.2 Interface Targets

An atomic target of the form (n,v) requires that a particular attribute value must appear in a request (to obtain a match). Such targets are little different conceptually from those defined in XACML 2.0 and other authorization languages and are, therefore, of limited novelty or interest here.[2]

[2] Targets in XACML 2.0 only consider subjects, objects and actions; targets in the draft XACML 3.0 do consider other types of attributes.

In contrast, targets of the form n, have not previously been seen in the literature on authorization languages (to the best of our knowledge). A target of the form n can be used to define a target that enforces a "request interface": a target of the form

$$\mathsf{opt}(n_1 \text{ and } n_2 \text{ and} \ldots \text{and } n_k),$$

for example, only matches a request that contains particular named attributes (corresponding to $n_1, \ldots, n_k$); the evaluation of a request that doesn't contain all the required attributes will evaluate to 0_{T} (because of the opt). In this way, we can construct a target that "guards" conventional subject-object-action policies and others that can respond to requests containing other types of attributes.

More complex "mixed" interfaces can also be constructed. An access control list is a type of access control data structure that is widely used in operating systems. The target for a policy used to represent an access control list for object *test.txt* would have the form

$$\mathsf{opt}((\texttt{object}, \mathit{test.txt}) \text{ and } \texttt{subject} \text{ and } \texttt{action}),$$

so that only requests that specify the desired object as well as including some subject and action would match.

3.3 On Functional Completeness

By way of motivation, we first observe that it might be useful to be able to define "conditional" interface targets, where the presence of one attribute in a request requires the presence of some other attribute. Suppose, for example, we have two attribute names n_1 and n_2. If a request doesn't contain attribute n_1 then the evaluation of the target should be 0_{T}. If, however, a request does contain n_1 then it must contain n_2. In other words, we have the following "match table", where the row headers indicate the values taken by the evaluation of n_1 and the column headers indicate the values taken by n_2.

	1_{T}	$\perp_{\mathrm{T}}$
1_{T}	1_{T}	$\perp_{\mathrm{T}}$
$\perp_{\mathrm{T}}$	0_{T}	0_{T}

By inspection of the match tables in Fig. 1, we see that the above table could be represented by the target $\sim x \tilde{\sqcap} y$, where x and y denote the evaluation of n_1 and n_2, respectively. However, the semantics of and are given by the operator $\sqcap$. Hence, it would be useful to demonstrate that our chosen target operators opt, not, or and and are functionally complete. In particular, we would prefer to define the interface target described above in terms of our existing operators, rather than having to introduce another type of target conjunction.

We now prove that for all n and any function $f : \mathsf{Dec}_{\mathrm{T}}^{n} \to \mathsf{Dec}_{\mathrm{T}}$, f can be constructed using the constants 1_{T}, 0_{T} and $\perp_{\mathrm{T}}$ and the operators opt, not and or. We obtain this property by proving that the three-valued logic expressed over the set $\{0_{\mathrm{T}}, 1_{\mathrm{T}}, \perp_{\mathrm{T}}\}$ and defined by the operators $\tilde{\sqcup}$, $\neg$ and $\sim$ is functionally complete, re-using a result of Jobe [10], stated below.

$\bullet$	3	2	1	E_1	E_2
3	3	2	1	3	1
2	2	2	1	1	2
1	1	1	1	2	3

(a) Over the set $\{3, 2, 1\}$

$\bullet$	1_T	$\bot_\mathrm{T}$	0_T	E_1	E_2
1_T	1_T	$\bot_\mathrm{T}$	0_T	1_T	0_T
$\bot_\mathrm{T}$	$\bot_\mathrm{T}$	$\bot_\mathrm{T}$	0_T	0_T	$\bot_\mathrm{T}$
0_T	0_T	0_T	0_T	$\bot_\mathrm{T}$	1_T

(b) Over the set $\{1_\mathrm{T}, \bot_\mathrm{T}, 0_\mathrm{T}\}$

Fig. 2. Jobe's 3-valued logic

Theorem 1 (Jobe 1962). *The three-valued logic E expressed over the set $\{1, 2, 3\}$ and defined by the operators $\bullet, E_1$ and E_2, given in Fig. 2(a), is functionally complete.*

Corollary 1. *The three-valued logic expressed over the set $\{0_\mathrm{T}, 1_\mathrm{T}, \bot_\mathrm{T}\}$ and defined by the operators $\tilde{\sqcup}$, $\neg$ and $\sim$ is functionally complete.*

Proof. We first define the operator $\tilde{\sqcap}$ from $\tilde{\sqcup}$ and $\neg$: for any $X_1, X_2 \in \mathsf{Dec}_\mathrm{T}$, $(X_1 \tilde{\sqcap} X_2) = \neg(\neg X_1 \tilde{\sqcup} \neg X_2)$[3].

We can clearly see from Fig. 2(b), that the operator $\tilde{\sqcap}$ is identical to $\bullet$ and $\neg$ is identical to E_1. Therefore, we only need to define a unary operator that swaps the values of 0_T and $\bot_\mathrm{T}$ while leaving 1_T unchanged. We write $\updownarrow$ to denote such an operator. The table below demonstrates that $\updownarrow X$ is equivalent to $(X \tilde{\sqcup} \bot_\mathrm{T}) \tilde{\sqcap} (\sim(X \tilde{\sqcup} \neg X))$.

X	$X \tilde{\sqcup} \bot_\mathrm{T}$	$\neg X$	$X \tilde{\sqcup} \neg X$	$\sim(X \tilde{\sqcup} \neg X)$	$\updownarrow X$
1_T	1_T	0_T	1_T	1_T	1_T
0_T	$\bot_\mathrm{T}$	1_T	1_T	1_T	$\bot_\mathrm{T}$
$\bot_\mathrm{T}$	$\bot_\mathrm{T}$	$\bot_\mathrm{T}$	$\bot_\mathrm{T}$	0_T	0_T

We can therefore conclude that the logic defined over the set $\{0_\mathrm{T}, 1_\mathrm{T}, \bot_\mathrm{T}\}$ by the operators $\tilde{\sqcup}, \neg$ and $\sim$ is functionally complete.

For instance, the operator and can be built directly from or and not, since we can define the operator $\sqcap$ from $\tilde{\sqcup}$ and $\neg$. Indeed, for any $x, y \in \mathsf{Dec}_\mathrm{T}$, we have the following equivalences:

$$x \sqcap y = (x \tilde{\sqcap} y) \tilde{\sqcup} ((x \tilde{\sqcap} \neg x) \tilde{\sqcup} (y \tilde{\sqcap} \neg y))$$
$$x \sqcup y = (x \tilde{\sqcup} y) \tilde{\sqcap} ((x \tilde{\sqcup} \neg x) \tilde{\sqcap} (y \tilde{\sqcup} \neg y))$$

We also have $x \dot{\sqcup} y = (x \tilde{\sqcup} (\sim y)) \tilde{\sqcap} ((\sim x) \tilde{\sqcup} y)$, where $\dot{\sqcup}$ is the supremum operator used to define the evaluation of an atomic target.

[3] Note that we also have the expected equivalence $(X_1 \tilde{\sqcup} X_2) = \neg(\neg X_1 \tilde{\sqcap} \neg X_2)$.

4 Policies

PTaCL policies are defined inductively. Let $d \in \{1_{\mathrm{P}}, 0_{\mathrm{P}}\}$, and let p, p_1 and p_2 be policies. Then

- d is a policy;
- $\mathsf{not}_{\mathrm{P}}\, p$ – the *negation* of policy p – is a policy, which returns 1_{P} if p returns 0_{P} and vice versa;
- $\mathsf{dbd}_{\mathrm{P}}\, p$ – the *deny-by-default* of policy p – is a policy, which returns 1_{P} if p returns 1_{P} and returns 0_{P} otherwise;
- $p_1\ \mathsf{and}_{\mathrm{P}}\ p_2$ – the *conjunction* of two policies p_1 and p_2 – is a policy;
- (t, p) – the *restriction* of policy p to a target t – is a policy.

We discuss policy evaluation in detail in Section 4.1.

A *policy tree* is a convenient way of visualizing a policy and can be constructed recursively from a policy. The policy d is represented as a tree comprising a single node. The policy $p_1\ \mathsf{and}_{\mathrm{P}}\ p_2$ is represented as a tree comprising a root node labelled $\mathsf{and}_{\mathrm{P}}$ and two child sub-trees representing p_1 and p_2. Policies of the form (t, p), $\mathsf{dbd}_{\mathrm{P}}\, p$ and $\mathsf{not}_{\mathrm{P}}\, p$ are represented as trees comprising a root node labelled t, $\mathsf{dbd}_{\mathrm{P}}$ and $\mathsf{not}_{\mathrm{P}}$, respectively, a single child sub-tree representing p. An illustrative policy tree representing the policy

$$\mathsf{dbd}_{\mathrm{P}}(t_5, \mathsf{not}_{\mathrm{P}}(t_3, (t_1, 1_{\mathrm{P}})\ \mathsf{and}_{\mathrm{P}}\ (t_2, 0_{\mathrm{P}}))\ \mathsf{and}_{\mathrm{P}}\ (t_4, 1_{\mathrm{P}}))$$

is shown in Fig. 3(a) (on page 402). To save space, we have "absorbed" the nodes labelled $\mathsf{and}_{\mathrm{P}}$ into their respective parents (t_3 and t_5).

4.1 Policy Evaluation

The evaluation of a policy with respect to a request q returns $\perp_{\mathrm{P}}$ if the policy is not applicable to the request: that is, the evaluation of the policy's target with respect to q returned 0_{T}. However, it may be the case that the evaluation of a target returns neither 1_{T} nor 0_{T}, instead returning $\perp_{\mathrm{T}}$. The possibility of target evaluation failing is considered in XACML [14] and in the work of Li *et al.* [12] and of Crampton and Huth [6]. The methods used to handle such failures assume that target evaluation failures arise because of unexpected failures in hardware, software or network connectivity and, accordingly, make a best effort to construct a conclusive decision for the request.

Our target language is expressly designed to support flexible request formats for open environments. As a result, our language explicitly includes the possibility that target evaluation may not be possible (if, for example, attributes are missing). Hence, target evaluation may fail, not because of "benign" failures, but because a user may withhold attributes in an attempt to force an error in target evaluation and thereby circumvent policy evaluation. Therefore, we must ensure that no advantage is gained by a malicious user who deliberately suppresses information when making an access request.[4]

[4] We also note the possibility that the user may not wish to divulge certain attributes when making an application request.

Our approach is to consider all possible decisions that might have arisen had target evaluation not failed. In other words, policy evaluation may return a set of decisions. We shall see that imposing appropriate restrictions on targets and using a "conservative" method of deriving a single decision from a set of decisions, will enable us to guarantee that a malicious user obtains no advantage by withholding attribute information.

We recall the operators $\neg$, $\sim$ and $\tilde{\sqcap}$ on Dec_T (as shown in Fig. 1) and define the same operators on $\mathsf{Dec}_\mathrm{P} = \{1_\mathrm{P}, 0_\mathrm{P}, \bot_\mathrm{P}\}$. We extend the unary operators to $X \subseteq \mathsf{Dec}_\mathrm{P}$, writing $\neg X$ to denote the set $\{\neg x : x \in X\}$ and $\sim X$ to denote the set $\{\sim x : x \in X\}$; and we extend $\tilde{\sqcap}$ on Dec_P to sets $X, Y \subseteq \mathsf{Dec}_\mathrm{P}$, writing $X \tilde{\sqcap} Y$ to denote the set $\{x \tilde{\sqcap} y : x \in X, y \in Y\}$.

Informally, the evaluation of targeted policy (t, p) for a request q proceeds in the following way.

1. If t evaluates to 1_T, we then inductively evaluate p (see below)
2. If t evaluates to 0_T, we return $\{\bot_\mathrm{P}\}$
3. Otherwise, we evaluate p and take the union of the resulting set of decisions with $\{\bot_\mathrm{P}\}$[5]

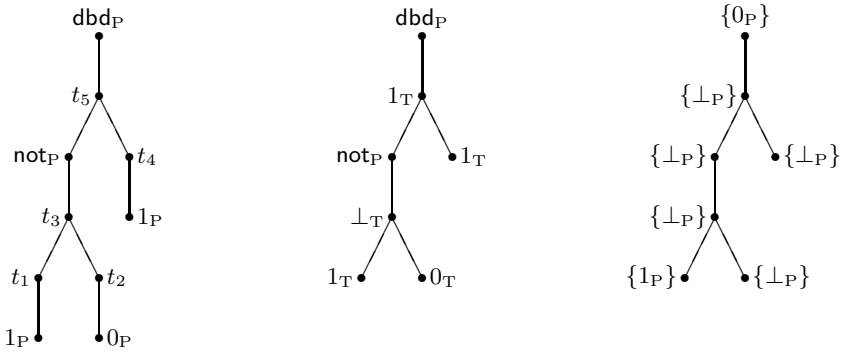
We write $[\![p]\!]_\mathrm{P}(q)$ to denote the evaluation of policy p with respect to a request q, where

$$
\begin{aligned}
&[\![d]\!]_\mathrm{P}(q) = \{d\}\,;\\
&[\![\mathsf{not}_\mathrm{P}\ p]\!]_\mathrm{P}(q) = \neg([\![p]\!]_\mathrm{P}(q));\\
&[\![\mathsf{dbd}_\mathrm{P}\ p]\!]_\mathrm{P}(q) = \sim([\![p]\!]_\mathrm{P}(q))\\
&[\![(p_1\ \mathsf{and}_\mathrm{P}\ p_2)]\!]_\mathrm{P}(q) = [\![p_1]\!]_\mathrm{P}(q) \tilde{\sqcap} [\![p_2]\!]_\mathrm{P}(q);\\
&[\![(t,p)]\!]_\mathrm{P}(q) = \begin{cases} [\![p]\!]_\mathrm{P}(q) & \text{if } [\![t]\!]_\mathrm{T}(q) = 1_\mathrm{T},\\ \{\bot_\mathrm{P}\} & \text{if } [\![t]\!]_\mathrm{T}(q) = 0_\mathrm{T},\\ \{\bot_\mathrm{P}\} \cup [\![p]\!]_\mathrm{P}(q) & \text{otherwise.}\end{cases}
\end{aligned}
$$

Consider the policy depicted in Fig. 3(a) and suppose that $[\![t_1]\!](q) = [\![t_4]\!](q) = [\![t_5]\!](q) = 1_\mathrm{T}$, $[\![t_2]\!](q) = 0_\mathrm{T}$ and $[\![t_3]\!](q) = \bot_\mathrm{T}$. The evaluation of this policy is shown in Fig. 3(c). Note that the evaluation of the sub-tree with root t_3 considers the union of two sets of decisions because $[\![t_3]\!](q) = \bot_\mathrm{T}$. Note also that the strong conjunction $\tilde{\sqcap}$ has the effect of preferring the $\bot_\mathrm{P}$ decision to the 1_P decision. For those familiar with previous related work, this may seem an unusual way in which to combine policy decisions. We discuss this in more detail in the next section and, in Section 4.3, we will discuss ways in which more familiar decision-combining operators can be defined. Finally, note that the policy does evaluate to a single decision (0_P) for this request, although there is no reason in general for this to occur. However, it is easy to establish the following result.

Lemma 2. *Let p be a policy whose policy tree contains targets $t_1, \ldots, t_k$ and let q be a request. If $[\![t_i]\!](q) \neq \bot_\mathrm{T}$ for all i, then $[\![p]\!](q) = \{x\}$ for some $x \in \mathsf{Dec}_\mathrm{P}$.*

[5] In other words, the evaluation of p in this case considers the decisions that would have been returned if the request had been applicable and if the request had not been applicable.



(a) Policy tree (b) Target evaluation (c) Policy evaluation

Fig. 3. Evaluating a PTaCL policy

In other words, if the applicability of all targets referenced by a policy can be determined for a request q, our evaluation semantics will return a unique authorization decision. The proof is a straightforward induction on the depth of the policy tree.

Finally, we note that the functional completeness for the target language also holds for our policy language, because opt_T and dbd_P have identical properties, as do not_T and not_P. However, it is also important to realize that the interpretation of $\perp_T$ and $\perp_P$ are quite different: the former indicates that the request supplied insufficient information to evaluate target applicability, whereas $\perp_P$ indicates that a policy is irrelevant to the evaluation of a request. Henceforth, we will omit the subscript from $[\![\cdot]\!]_P$ and the PTL operators, although, for clarity, we will retain the subscripts on decisions.

4.2 On the Non-monotonicity of Targets

The language we use for targets and the way in which targets are evaluated means that, for some target t, there may exist requests q and q' such that $q' \subseteq q$, $[\![t]\!](q') = 0_T$ and $[\![t]\!](q) = 1_T$. This feature of the language means that withholding attributes may provide some advantage to a malicious user: if we have a policy $p = (t, p')$ such that $[\![p']\!](q) = 0_P$, and $[\![t]\!](q) = 1_T$, then $[\![p]\!](q) = 0_P$; if, however, $[\![t]\!](q) = 0_T$, then $[\![p]\!](q) = \perp_P$. In other words, it might be possible for a malicious user to turn a 0_P decision into a $\perp_P$ decision by suppressing certain attributes. For brevity, we refer to this as the non-monotonicity of targets. Hence, we might reasonably regard $\perp_P$ as a potentially dangerous policy decision. (This view of $\perp_P$ is quite different from the interpretation used by other policy languages and algebras.) It is this view that informs our use of $\tilde{\sqcap}$ to combine policy decisions, which means that $\perp_P \tilde{\sqcap} 1_P$ is defined to be $\perp_P$ rather than 1_P.

Similarly, a user can force a target to evaluate to $\perp_T$ (rather than 0_T or 1_T) by withholding attributes. It is for this reason, that policy evaluation considers

the possibility that a target might have been matched or not matched when target evaluation returns $\bot_{\mathrm{T}}$.

Following from the above discussion, we would like to prove a result of the form: *Let p be a policy whose policy tree contains targets $t_1, \ldots, t_k$ and let q be a request. Then for any $q' \subseteq q$, $[\![p]\!](q) \subseteq [\![p]\!](q')$.* Informally, this result states that if a request contains less information, then the result of evaluating the policy is more uncertain. Then the authorization decision point can have a decision-set "resolution strategy" that returns a single final decision. Such a strategy should be "conservative" in the sense that the larger decision sets should be treated with more caution. The obvious strategy of this nature is: for $X \subseteq \mathsf{Dec}_{\mathrm{P}}$, we return 1_{P} if $X = \{1_{\mathrm{P}}\}$ and 0_{P} otherwise.

However, it is easy to see that the above result does not hold, because of the functional completeness of our target language. In particular, we can create an operator $\oplus$ such that $\bot_{\mathrm{T}} \oplus \bot_{\mathrm{T}} = 1_{\mathrm{T}}$ and $1_{\mathrm{T}} \oplus \bot_{\mathrm{T}} = \bot_{\mathrm{T}}$. Now consider the target $t = (n_1, v_1) \oplus (n_2, v_2)$, and the requests $q_1 = \{(n_1, v_1)\}$ and $q_2 = \{\}$. Then

$$[\![t]\!](q_1) = 1_{\mathrm{T}} \oplus \bot_{\mathrm{T}} = \bot_{\mathrm{T}} \quad \text{and} \quad [\![t]\!](q_2) = \bot_{\mathrm{T}} \oplus \bot_{\mathrm{T}} = 1_{\mathrm{T}}.$$

Now consider the policy $p = (t, 1_{\mathrm{P}})$: we have $[\![p]\!](q_1) = \{\bot_{\mathrm{P}}, 1_{\mathrm{P}}\}$ and $[\![p]\!](q_2) = \{1_{\mathrm{P}}\}$, providing a counter-example to the desired result. In other words, there are good reasons to restrict our target language so that only "well-behaved" targets can be defined. Specifically, we would like to restrict our target language so that all targets have the following property:

Definition 3. *A target t is* monotonic *if for all requests q and for every $q' \subseteq q$, $[\![t]\!](q') \in \{\bot_{\mathrm{T}}, [\![t]\!](q)\}$.*

Then we have the following result (all the proofs of this paper are given in [7] and have been encoded in the proof assistant Isabelle/Isar[6]).

Theorem 4. *Let p be a policy whose policy tree contains* monotonic *targets $t_1, \ldots, t_k$ and let q be a request. Then for any $q' \subseteq q$, $[\![p]\!](q) \subseteq [\![p]\!](q')$.*

The obvious questions to ask now are: Which of our target operators are monotonic? And does composition of monotonic target operators preserve monotonicity?

We say that an operator is monotonic if, given monotonic targets as inputs, it returns a monotonic target. We prove in [7] that the operators not, and and or are monotonic, as well as the operators corresponding to $\tilde{\sqcap}$ and $\sqcup$. However, the operator opt is not monotonic, since it can transform a $\bot_{\mathrm{T}}$ into a 0_{T}.

Unfortunately (and somewhat unexpectedly), an atomic target is not, in general, monotonic. To see this, note that a request can contain several pairs with the same attribute name. (A request might, for example, enumerate all the roles with which the requester is associated.) Removing one occurrence from this set of pairs can change the evaluation of the request from 1_{T} to 0_{T}. This situation corresponds to a partial hiding of attribute values: that is, the ability for a user or an attribute server to remove only some values for a given attribute. In practice,

[6] `http://isg.rhul.ac.uk/~jason/isabelle/ptacl.thy`

such a situation is quite hard to detect and to prevent. However, let us assume that an attribute server works in an "all-or-nothing mode": that is, either it returns all the values for a given attribute, or none. With this assumption, for two requests q and q' such that $q' \subseteq q$ and for any attribute name n such that $(n, v) \in q'$ and $(n, v') \in q$, then $(n, v') \in q'$. With such an assumption, it is easy to see that any atomic target is monotonic, and it follows that any target built using the operators and, or and not is monotonic.

Such an assumption might not always hold, in particular when there is little control over the attribute servers. Therefore, we now consider an alternative, weaker notion of monotonicity, defined below.

Definition 5. *A target t is* weakly monotonic *if for all requests q and for every $q' \subseteq q$, $[\![t]\!](q') \preccurlyeq [\![t]\!](q)$, where we define $\perp_{\mathrm{T}} \prec 0_{\mathrm{T}} \prec 1_{\mathrm{T}}$.*

The operators $\sim, \sqcap, \sqcup$ and $\tilde{\sqcup}$ preserve the weak monotonicity, as proven in [7], but the operators $\neg$ and $\tilde{\sqcap}$ do not. Moreover, since any atomic target is clearly weakly monotonic, any target built using any combination from the operators $\sim, \sqcap, \sqcup$ and $\tilde{\sqcup}$ is also weakly monotonic. Although we cannot prove a result as strong as Theorem 4, we can prove the following result (the proof of which can be found in [7]).

Theorem 6. *Let p be a policy whose policy tree contains* weakly monotonic *targets $t_1, \ldots, t_k$ and let q be a request.*

1. *If p is constructed from the operators* not *and* and*, then for any $q' \subseteq q$, if $[\![p]\!](q') = \{d\}$, with $d \in \{1_{\mathrm{P}}, 0_{\mathrm{P}}\}$, then $[\![p]\!](q) = [\![p]\!](q')$.*
2. *If p is constructed from the operators* dbd *and* and*, then for any $q' \subseteq q$, if $[\![p]\!](q') = \{1_{\mathrm{P}}\}$, then $[\![p]\!](q) = \{1_{\mathrm{P}}\}$.*

One consequence of Theorem 6 is that if a partial request is allowed, then the full request would have been allowed too, and therefore an attacker has no advantage in hiding some attribute values. However, this result requires a "conservative" resolution strategy: that is, request q is only allowed if and only if $[\![p]\!](q) = \{1_{\mathrm{P}}\}$.

4.3 Decision Operators

We now discuss other ways in which decisions from sub-policies might be combined. Following Crampton and Huth [6], we restrict attention to *idempotent* and *well-behaved* decision operators.

Definition 7. *Let $\oplus : \mathsf{Dec}_{\mathrm{P}} \times \mathsf{Dec}_{\mathrm{P}} \to \mathsf{Dec}_{\mathrm{P}}$ be a decision operator.*

- *If $x \oplus x = x$ for all $x \in \mathsf{Dec}_{\mathrm{P}}$, then we say $\oplus$ is* idempotent.
- *If $x \oplus \perp_{\mathrm{P}} = x = \perp_{\mathrm{P}} \oplus x$ for all $x \in \mathsf{Dec}_{\mathrm{P}}$, then we say $\oplus$ is a* $\cup$-operator.
- *If $x \oplus \perp_{\mathrm{P}} = \perp_{\mathrm{P}} = \perp_{\mathrm{P}} \oplus x$ for all $x \in \mathsf{Dec}_{\mathrm{P}}$, then we say $\oplus$ is an* $\cap$-operator.
- *We say $\oplus$ is* well-behaved *if it is either a $\cup$- or an $\cap$-operator.*

Informally, a $\cup$-operator ignores policies that evaluate to $\perp_{\mathrm{P}}$ by returning a conclusive decision (that is, a decision that belongs to $\{1_{\mathrm{P}}, 0_{\mathrm{P}}\}$) if either operand returns a conclusive decision. XACML, for example, assumes that all operators are $\cup$-operators. In contrast, a $\cap$-operator only returns a conclusive decision if both arguments are conclusive decisions. An operator of this nature is used by Bonatti *et al.* in their policy algebra [4].

Intuitively, it seems reasonable to assume that a policy decision operator is idempotent: if two policies return the same decision d, then we would expect that the composition of those policies would also return d. An idempotent, well-behaved decision operator is uniquely defined by the choices of $x \oplus \perp_{\mathrm{P}}$, $1_{\mathrm{P}} \oplus 0_{\mathrm{P}}$ and $0_{\mathrm{P}} \oplus 1_{\mathrm{P}}$: the remaining values are fixed because the operator is idempotent and well-behaved (as shown in Fig. 4 for an idempotent $\cup$-operator $\oplus$).

If we assume that $\oplus$ is commutative, then there are only three choices for an idempotent $\cup$-operator (and three choices for an idempotent $\cap$-operator). And if we assume that $1_{\mathrm{P}} \oplus 0_{\mathrm{P}} \in \{1_{\mathrm{P}}, 0_{\mathrm{P}}\}$, then there are only two choices for a commutative, idempotent $\cup$-operator; both these operators are shown in Fig. 4, labeled as $\mathsf{and}_{\cup}$ and $\mathsf{or}_{\cup}$. Analogous operators $\mathsf{and}_{\cap}$ and $\mathsf{or}_{\cap}$ can be defined by making the obvious adjustments to the bottom row and rightmost column of the tables for $\mathsf{and}_{\cup}$ and $\mathsf{or}_{\cup}$, respectively.

The operators $\mathsf{and}_{\cup}$ and $\mathsf{and}_{\cap}$ are rather similar to logical conjunction, while $\mathsf{or}_{\cup}$ and $\mathsf{or}_{\cap}$ are rather similar to logical disjunction, respectively. Our decision operators play a similar role to the conflict resolution strategies or policy-combining algorithms used in policy algebras and XACML. Such strategies are used to resolve discrepancies in the results returned by different sub-policies. In particular, $\mathsf{and}_{\cup}$ has the same effect as the "deny-overrides" conflict resolution strategy: namely, if one sub-policy returns 0_{P}, then the combined decision is 0_{P}. Similarly, $\mathsf{or}_{\cup}$ has the same effect as the "allow-overrides" strategy.

The most widely used non-commutative conflict resolution strategy is "first-applicable", which we denote by $\rhd$. The operator $\rhd$ is defined in Fig. 4(d): note, in particular, $1_{\mathrm{P}} \rhd 0_{\mathrm{P}} = 1_{\mathrm{P}}$ and $0_{\mathrm{P}} \rhd 1_{\mathrm{P}} = 0_{\mathrm{P}}$.[7] The first-applicable operator is commonly used in firewall rulesets as well as in policy algebras and XACML. The other idempotent, well-behaved, non-commutative operator such that $1_{\mathrm{P}} \oplus 0_{\mathrm{P}} \in \{1_{\mathrm{P}}, 0_{\mathrm{P}}\}$ and $0_{\mathrm{P}} \oplus 1_{\mathrm{P}} \in \{1_{\mathrm{P}}, 0_{\mathrm{P}}\}$ is what might be called "last-applicable", denoted by $\lhd$, where $x \lhd y = y$ if $y \in \{1_{\mathrm{P}}, 0_{\mathrm{P}}\}$ and is equal to x otherwise. This operator does not appear to be widely supported or used.

We now show how to define the operators $\mathsf{or}_{\cap}$, $\mathsf{and}_{\cap}$, $\mathsf{or}_{\cup}$, $\mathsf{and}_{\cup}$ and $\rhd$ from the PTL operators not, dbd and and. Since the logic $(\{1_{\mathrm{P}}, 0_{\mathrm{P}}, \perp_{\mathrm{P}}\}, \mathsf{not}, \mathsf{dbd}, \mathsf{and})$ is functionally complete, we can directly reuse the definitions of the operators given in Fig. 1. Clearly, $\mathsf{or}_{\cap}$ and $\mathsf{and}_{\cap}$ are directly given by $\sqcup$ and $\sqcap$, respectively. Moreover, the operator $\mathsf{or}_{\cup}$ corresponds to the supremum operator over the total order $1_{\mathrm{P}} > 0_{\mathrm{P}} > \perp_{\mathrm{P}}$, so we can re-use the operator $\dot{\sqcup}$ defined in Section 3.3. The operator $\mathsf{and}_{\cup}$ is defined as follows:

$$x \,\mathsf{and}_{\cup}\, y = \mathsf{not}((\mathsf{not}\, x)\, \mathsf{or}_{\cup}\, (\mathsf{not}\, y))$$

[7] Note that a first-applicable $\cap$-operator is vacuous, as it would be equivalent to a unary, identity operator.

$\oplus$	1_P	0_P	$\bot_P$
1_P	1_P	x	1_P
0_P	y	0_P	0_P
$\bot_P$	1_P	0_P	$\bot_P$

(a) Idempotent

$\mathsf{and}_\cup$	1_P	0_P	$\bot_P$
1_P	1_P	0_P	1_P
0_P	0_P	0_P	0_P
$\bot_P$	1_P	0_P	$\bot_P$

(b) Conjunction

$\mathsf{or}_\cup$	1_P	0_P	$\bot_P$
1_P	1_P	1_P	1_P
0_P	1_P	0_P	0_P
$\bot_P$	1_P	0_P	$\bot_P$

(c) Disjunction

$\rhd$	1_P	0_P	$\bot_P$
1_P	1_P	1_P	1_P
0_P	0_P	0_P	0_P
$\bot_P$	1_P	0_P	$\bot_P$

(d) First-applicable

Fig. 4. Decision tables for idempotent $\cup$-operators on $\{1_P, 0_P, \bot_P\}$

In order to define the operator $\rhd$, we first introduce the operator abd ("allow-by-default"), which transforms $\bot_P$ into 1_P, and is defined by $\mathsf{abd}\, x = \mathsf{not}(\mathsf{dbd}(\mathsf{not}\, x))$. The definition of $\rhd$ is then given by:

$$x \rhd y = (\mathsf{abd}(x \,\tilde{\sqcup}\, (\mathsf{not}\, x))) \,\tilde{\sqcap}\, (x \,\mathsf{or}_\cup\, y)$$

Finally, $x \lhd y$ is equivalent to $y \rhd x$. Henceforth, we will use the operators defined above as syntactic sugar. Notice that our definitions of $\mathsf{or}_\cup$, $\mathsf{and}_\cup$ and $\rhd$ all require the three PTL operators for their construction. Hence, a policy containing the standard XACML operators does not satisfy the requirements of Theorem 6, so we need to rely on the all-or-nothing assumption.

Finally, we note that the operators and, $\mathsf{and}_\cup$ and $\mathsf{and}_\cap$ can be regarded as defining a greatest lower bound operator for suitable choices of ordering on Dec_P; similarly $\mathsf{or}_\cup$ and $\mathsf{or}_\cap$ define least upper bound operators. These orderings are summarized in Table 1.

Table 1. Decision operators and orderings on Dec_P

Operator	Ordering
and	$0_P < \bot_P < 1_P$
$\mathsf{and}_\cup$	$0_P < 1_P < \bot_P$
$\mathsf{and}_\cap$	$\bot_P < 0_P < 1_P$
$\mathsf{or}_\cup$	$\bot_P < 0_P < 1_P$
$\mathsf{or}_\cap$	$0_P < 1_P < \bot_P$

The fact that each of the orderings is a total order means that and, $\mathsf{and}_\cup$ and $\mathsf{and}_\cap$ take the minimum of their operands, while $\mathsf{or}_\cup$ and $\mathsf{or}_\cap$ take the maximum of their operands. This, in turn, means that all four operators can be extended to n-ary operators (for any natural number $n > 1$).

5 Related Work

It is important to note that PTaCL is neither intended to fix XACML nor to provide formal semantics for XACML policy evaluation. Rather, PTaCL is a

language that seeks to provide rigorous, alternative solutions to the same problems that motivated the development of XACML. Our work is also influenced by the work of Li *et al.* [12] and of Crampton and Huth [6] on using a set of decisions, rather than a single decision, to define the result of policy evaluation.

Although there is a substantial body of work on policy specification [1,4,5,13,17], this prior work assumes a very restricted format for access requests and targets. To the best of our knowledge, there is no previous work on a formal language for target specification and evaluation, let alone the consideration of missing attributes names. Both the ratified standard XACML 2.0 [14] and the draft XACML 3.0 [15], acknowledge that attributes may be missing from a request. However, the treatment of target evaluation in such circumstances is, like much of the XACML standard, rather informal. Moreover, the XACML target syntax is unnecessarily complicated and does not support interface targets. Finally, the XACML target syntax only provides operators that are equivalent to the strong conjunction and strong disjunction (in the 3-value Kleene logic), thereby limiting the expressive power of XACML. On the other hand, the functional completeness of PTL means that any XACML target can be represented in PTL.

The work on policy algebras varies in the operators that are supported, the set of decisions that can arise as a result of policy evaluation, and the extent to which policy evaluation can cope with failures in target evaluation. Ni *et al.*, for example, provide a functional complete policy algebra [13], where policy evaluation returns a single decision from the set $\{1_{\mathrm{P}}, 0_{\mathrm{P}}, \bot_{\mathrm{P}}\}$. The functional completeness of PCL means that we can express any operators that we might wish to. In particular, we can express all XACML policy-combining algorithms. Structurally, our atomic policies correspond to rules in XACML, while our policy trees correspond to policies and policy sets. Crampton and Huth [6] extend the work of Li *et al.* on policy evaluation in the presence of target evaluation failure [12], where policy evaluation returns a set of decisions. Our treatment of policy evaluation is rather similar to this earlier work, although the way in which we resolve a set of decisions to a single decision that is enforced by the AEF is completely different, due to the suspicion with which we choose to treat the $\bot_{\mathrm{P}}$ decision.

An important contribution of this paper is the recognition that providing support for attribute-based access control and greater freedom for request formats leads to the potential for attribute hiding by malicious users. By manipulating requests in this way, it may be possible to circumvent the expected or intended policy semantics. Existing work that supports attribute-based access control, such as XACML 3.0 and that of Rao *et al.* [16], does not consider such possibilities and hence may be vulnerable to "attribute-hiding attacks". Consider, for example, the PTL policy $p = (1_{\mathrm{P}}\ \mathsf{and}_{\cup}\ ((n, v), 0_{\mathrm{P}}))$ – which corresponds to an XACML policy with two rules combined using the deny-overrides operator – and two requests $q = \{(n, v), (n, v')\}$ and $q' = \{(n, v')\}$. Then $[\![p]\!](q) = 0_{\mathrm{P}}$ while $[\![p]\!](q') = 1_{\mathrm{P}}$: that is, by hiding some information, a more favorable answer is obtained. Theorem 6 suggests that such behavior is to be expected because we require all three PTL operators to represent $\mathsf{and}_{\cup}$.

6 Concluding Remarks

Attribute-based access control, rather than the traditional identity-based access control that is deployed extensively in closed systems, is likely to become increasingly important in loosely coupled and open computing environments. This paper introduces PTaCL, an expressive language for the definition of attribute-based authorization policies. PTaCL can represent all commonly used policy composition operators (indeed it can represent any desired operator) and, to the best of our knowledge, PTaCL is the first language with a concise syntax for policy targets and a precise semantics for their evaluation.

Nevertheless, PTaCL is rather simple syntactically, which enables us to identify and propose solutions to the problem of attribute hiding. Such an issue is problematic in the context of open and distributed systems, and is not addressed in the literature, which define composition operators to favor conclusive decisions over a not-applicable decision. Having identified the problem, we propose two approaches to address this issue, formally justifying each of them: either forbidding optional targets, assuming the attribute servers to work in an "all-or-nothing mode" and adopting a conservative evaluation; or constraining more strictly the definition of the targets and the definition of the policies. The second approach does not make any assumption about the behavior of the attribute servers, but the standard policy composition operators can no longer be used. We propose other operators that are resilient to attribute hiding and differ from the standard ones in the way in which they handle the not-applicable decision. These "new" operators actually correspond to the strong conjunction and strong disjunction defined in the original Kleene three-valued logic.

There are many opportunities for future work. Clearly, when the evaluation of a request returns more than one decision, it implies that some attributes are missing in the request, and PTaCL should be extended in order for the set of the decisions to also indicate which attributes are missing. Hence, the entity in charge of collecting the attributes, for instance the Context Handler in the XACML architecture. Hence, a useful extension to the operational semantics of PTaCL would be to extend the return type of PCL so that the response includes a list of missing attribute names. PCL can be similarly extended in order to support obligations, that can be returned in addition to a set of decisions (as in XACML).

These extensions naturally lead to the problem of understanding and formalizing the complete access control architecture, and in particular to the question of *attribute privacy*. Indeed, in practice, a reason for a missing attribute can be because the source responsible for providing its value considered that this value was too sensitive to be shared. In such a case, the evaluation of the policy, or part of it, needs to be delegated to the attribute source. However, the possible presence of multiple, sensitive and conflicting sources makes it a non-trivial problem to solve. We believe that by completely formalizing the notion of attribute and its treatment by the policy decision point, PTaCL paves the way to address the problem of attribute privacy.

References

1. Backes, M., Dürmuth, M., Steinwandt, R.: An Algebra for Composing Enterprise Privacy Policies. In: Samarati, P., Ryan, P.Y.A., Gollmann, D., Molva, R. (eds.) ESORICS 2004. LNCS, vol. 3193, pp. 33–52. Springer, Heidelberg (2004)
2. Bell, D., LaPadula, L.: Secure computer systems: Unified exposition and Multics interpretation. Technical Report MTR-2997, Mitre Corporation (1976)
3. Bertino, E., Castano, S., Ferrari, E.: Author-$\mathcal{X}$: A comprehensive system for securing XML documents. IEEE Internet Computing 5(3), 21–31 (2001)
4. Bonatti, P., De Capitani Di Vimercati, S., Samarati, P.: An algebra for composing access control policies. ACM Transactions on Information and System Security 5(1), 1–35 (2002)
5. Bruns, G., Huth, M.: Access-control policies via Belnap logic: Effective and efficient composition and analysis. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium, pp. 163–176 (2008)
6. Crampton, J., Huth, M.: An Authorization Framework Resilient to Policy Evaluation Failures. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 472–487. Springer, Heidelberg (2010)
7. Crampton, J., Morisset, C.: Ptacl: A language for attribute-based access control in open systems. CoRR, abs/1111.5767 (2011), `http://arxiv.org/abs/1111.5767`
8. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P.: A fine-grained access control system for XML documents. ACM Transactions on Information and System Security 5(2), 169–202 (2002)
9. Harrison, M., Ruzzo, W., Ullman, J.: Protection in operating systems. Communications of the ACM 19(8), 461–471 (1976)
10. Jobe, W.: Functional completeness and canonical forms in many-valued logics. Journal of Symbolic Logic 27(4), 409–422 (1962)
11. Kleene, S.: Introduction to Metamathematics. D. Van Nostrand, Princeton, NJ (1950)
12. Li, N., Wang, Q., Qardaji, W., Bertino, E., Rao, P., Lobo, J., Lin, D.: Access control policy combining: Theory meets practice. In: Proceedings of 14th ACM Symposium on Access Control Models and Technologies, pp. 135–144 (2009)
13. Ni, Q., Bertino, E., Lobo, J.: D-algebra for composing access control policy decisions. In: Proceedings of 2009 ACM Symposium on Information, Computer and Communications Security, pp. 298–309 (2009)
14. OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0, OASIS Committee Specification (Tim Moses, editor) (2005)
15. OASIS. eXtensible Access Control Markup Language (XACML) Version 3.0, OASIS Committee Specification 01 (Erik Rissanen, editor) (2010)
16. Rao, P., Lin, D., Bertino, E., Li, N., Lobo, J.: An algebra for fine-grained integration of XACML policies. In: Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, pp. 63–72. ACM, New York (2009)
17. Wijesekera, D., Jajodia, S.: A propositional policy algebra for access control. ACM Transactions on Information and System Security 6(2), 235–286 (2003)