An Ontological Framework for Reasoning About Relations Between Complex Access Control Policies in Cloud Environments

Simeon Veloudis, Iraklis Paraskakis and Christos Petsos

South East European Research Centre (SEERC)
The University of Sheffield, International Faculty CITY College
Thessaloniki, Greece

Keywords: Context-Aware Security, Privacy, Access Control, Inter-Policy Relations, Ontological Representation, Semantic Inferencing, Reasoning

Abstract: By embracing the cloud computing paradigm enterprises are able to realise significant cost savings whilst boosting their agility and productivity. Yet, due mainly to security and privacy concerns, many enterprises are reluctant to migrate the storage and processing of their critical assets to the cloud. One way to alleviate these concerns, hence bolster the adoption of cloud computing, is to infuse suitable *access control policies* in cloud services. Nevertheless, the complexity inherent in such policies, stemming from the dynamic nature of cloud environments, calls for a framework capable of providing assurances with respect to the *effectiveness* of these policies. The work presented in this paper elaborates on such a framework. In particular, it proposes an approach for generically checking potential *subsumption* relations between access control policies that incorporate the contextual knowledge that characterises an access request and which needs to be taken into account for granting, or denying, the request. The proposed framework is expressed ontologically hence enabling automated reasoning, through semantic inferencing, about policy subsumption.

1 INTRODUCTION

Cloud computing signifies a shift towards service-based architectures that offer a theoretically boundless scalability and a flexible pay-per-use model (Liu et al., 2011). Such a shift induces significant advantages for enterprises in terms of cost, flexibility and business agility. In particular, it enables inherently heterogeneous stakeholders, ranging from SMEs to large retailers and healthcare institutions, to realise significant cost savings by migrating their data and applications to servers that are under the control of third-party cloud providers. However, relinquishing control of —oftentimes critical— corporate assets naturally raises significant *security* and *privacy* concerns that deter, in general, stakeholders from embracing the cloud paradigm (CSA, 2015).

One way to alleviate these concerns, hence reinforce the adoption of cloud computing, is to infuse adequate *access control policies* into the applications through which critical assets are accessed in the cloud (Veloudis and Paraskakis, 2016). Nevertheless, the inherently dynamic nature of cloud environments calls for policies that are able to incorporate a potentially complex body of *contextual knowledge* (Veloudis et al., 2017b). As an example, consider a policy whereby a particular entity (say s) is allowed to read a sensitive data object (say o) only when the following conditions hold conjunctively: (i) o resides in a data centre in Ireland; (ii) s resides in a specific geographical area, say the EU, or the request originates from a particular subnet; (iii) the request is received during a prescribed time interval; (iv) the request is issued over a particular type of connection (e.g. one that utilises a certain cipher suite).

We argue that, for stakeholders to be able to rely on such complex policies for the protection of their sensitive assets, a generic *governance* framework that is capable of providing assurances about the *effectiveness* of the policies is required (Veloudis and Paraskakis, 2016). Our work, conducted as part of the PaaSword project (Paa, 2015), provides such a framework. In particular, it provides a governance mechanism that draws upon a *semantic representation* of policies, one that captures *ontologically* the various *knowledge artefacts* comprised in policies, and therefore unravels the expression of policies from the actual code of the applications into which they are infused. This renders our mechanism capable ~~of providing the following seminal capabilities. Firstly, it enables automated reasoning about the *correctness* of policies by constraining which particular knowledge~~

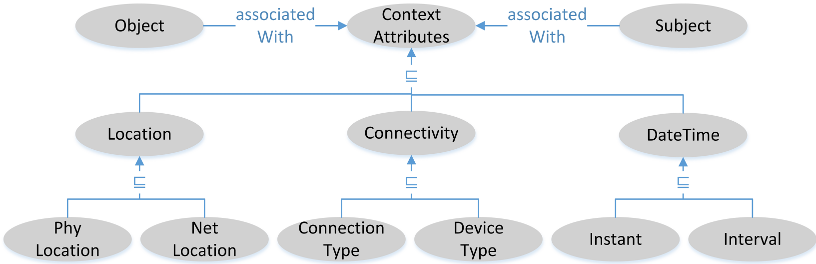

Figure 1: Simplified view of the CM.

~~artefacts may, must, or may not be encoded in the policies, as well as by determining the allowable value ranges that such knowledge artefacts may assume inside a policy. Secondly, it enables~~ automated reasoning about potential *inter-policy relations*, such as subsumption and contradiction, that may affect the effectiveness of the policies. ~~Thirdly, it enables the performance of *policy lifecycle* actions, such as policy updates and retirements, in a controlled and rule-based manner, i.e. in a manner that does not undermine the effectiveness of the policies.~~

~~The first and third capabilities above have been reported in (Veloudis et al., 2017a) and (Veloudis et al., 2019) respectively;~~ this paper ~~focuses on the second capability. More specifically, it~~ presents an approach to reasoning about *subsumption* relations between complex context-aware access control policies. The approach draws upon the diffused XACML standard (XAC, 2013), whilst it performs inter-policy comparisons, aiming at unveiling subsumption relations, on the basis of a suitable semantic characterisation of all those access requests that may be permitted, or denied, by the policies. One of the main benefits of our approach is that, by modelling policies ontologically —using OWL (OWL, 2004)— we can delegate reasoning about policy subsumption to off-the-shelf DL reasoners. In addition, our approach enables —by virtue of *semantic inferencing*— the generation of new *knowledge artefacts* which potentially allows the detection of subsumption relations in situations in which the knowledge artefacts encoded in the policies are syntactically incomparable. ~~Assume, for instance the example access control policy above and suppose a second policy whereby the entity *s* is allowed to read the data object *o* when *o* resides in a data centre in Ireland and *s* resides in, say, the city of Athens. This second policy is subsumed by the first as semantic inferencing allows us to assert that the city of Athens is indeed located in the EU and hence the body of contextual knowledge associated with the second policy implies the corresponding body associated with the first.~~

The rest of this paper is structured as follows. Section 2 outlines the *Context Model* — a customisable ontological model underpinning our framework ~~that encompasses concepts, and their interrelations, which represent the various knowledge artefacts embodied in policies~~. Section 3 proposes an ontological representation of an XACML-based model for policies and provides an ontological representation for access requests. Section 4 articulates a semantic characterisation of the conditions under which a policy is enforceable upon an access request. Section 5 ~~capitalises on this semantic characterisation and~~ outlines an approach to determining subsumption between policies and policy sets. Finally, Section 6 presents related work and Section 7 conclusions and future work.

2 REPRESENTING KNOWLEDGE ARTEFACTS

The Context Model (CM) proposed in (Verginadis et al., 2015) provides a suitable ontological framework for the representation of the various knowledge artefacts, or *attributes*, associated with access requests, hence with access control policies. At the core of the model is the class[1] *ContextAttributes* which, as its name suggests, encompasses concepts that represent *contextual attributes*; these are depicted in Figure 1 and briefly outlined in ~~Table 1~~ — for more details, the interested is referred to (Verginadis et al., 2015; Veloudis et al., 2016). Other attributes include the *subjects* and *objects* of requests which are represented as instances of the classes *Subject* and *Object* respectively, whilst they are associated with the contextual attributes that characterise them through the object property *associatedWith* (see Figure 1). A request also incorporates an *action* (e.g. a read, write or execute action) which is represented as an instance of

[1]The terms ‘class’ and ‘concept’ are used interchangeably. All concepts and properties of the CM belong to the *pcm* namespace (Verginadis et al., 2015); in order to avoid notational clutter, we omit the *pcm* prefix from concept and property identifiers.

Table 1: Context attributes.

Concept	Description
Location	Describes a physical or network location that characterises the subject or object of a request; to this end, it includes the classes *PhyLocation* and *NetLocation*. The former comprises such concepts as *Address*, *Point* and *Area* that further specify a physical location; analogous concepts are comprised by the latter (see (Verginadis et al., 2015) for more details).
DateTime	Describes a chronological point, or interval, that may characterise the action associated with a request; it incorporates the concepts *Instant* and *Interval*, along with a multitude of relevant data properties (not shown in Figure 1) that further specify the concepts (see (Verginadis et al., 2015) for more details).
Connectivity	Captures information pertaining to the connection (e.g. which cipher suite is utilised), or type of device (e.g. desktop, smart phone, tablet, etc.), that is used for accessing sensitive assets.

the class *Action* (not shown in Figure 1).

3 ACCESS CONTROL POLICIES AND REQUESTS

As already mentioned, the dynamic nature of cloud environments calls for access control policies that are able to incorporate the contextual knowledge pertaining to access requests. *Attribute-Based Access Control* (*ABAC*) policies (Hu et al., 2014), due to their inherent generality stemming from their reliance on the generic concept of an *attribute*, are deemed particularly suitable for capturing such knowledge (Veloudis and Paraskakis, 2016) and are therefore adopted in our work. The purpose of this section is to present a model for the representation of ABAC policies. In particular, Section 3.1 provides an informal account of an XACML-based policy model and Section 3.2 provides an ontological expression of this model; finally, Section 3.3 outlines an ontological representation of access requests.

3.1 ABAC Policy Model

Following the XACML standard (XAC, 2013), an ABAC policy comprises one or more *ABAC rules*. Rules are the most elementary structural elements and the basic building blocks of policies. In this respect, they are the carriers of the core logic that dwells in policies. Each ABAC rule consists of an *antecedent* and a *consequent*. The former articulates a (*pre-*)*condition* (or '*target*' in the XACML jargon) that must be satisfied in order for the rule to be *enforceable*. More specifically, it incorporates a set of relevant knowledge artefacts, its so-called *attributes*, that are captured in terms of the concepts introduced by the CM, and whose values need to be taken into account when deciding whether to permit, or deny, a particular request. On the other hand, the consequent of an ABAC rule captures the *decision* associated with the rule; such a decision invariably resolves to either a '*permit*' or a '*deny*'.

An ABAC policy is also invariably linked to a *rule-combining algorithm* (XAC, 2013) that determines which one of the policy's rules (if any) is to be applied when responding to an access request. It follows that, for each access request, an ABAC policy resolves to at most one of its constituent rules; a policy that does not resolve to any of its constituent rules for a particular request is considered '*NotApplicable*' for that request.

ABAC policies may also be grouped into ABAC *policy sets* which potentially comprise, in addition to policies, other policy sets as well. Each ABAC policy set is linked to a *policy-combining algorithm* (XAC, 2013) that determines which one of its elements (if any) is to be enforced when responding to a particular access request. It follows that, for each access request, an ABAC policy set resolves to at most one of its elements (either a policy or a nested policy set); a policy set that does not resolve to any of its elements for a particular request is considered '*NotApplicable*' for that request.

3.2 Ontological Representation

Our ontological representation of the ABAC policy model comprises four main concepts[2]: *Policy*, *Rule*, *PolicySet* and *CombiningAlg* (see Figure 2). Evidently, instances of these concepts represent, respec-

[2]All concepts and properties of the ABAC policy model belong to the *pac* namespace (Veloudis and Paraskakis, 2015a); in order to avoid notational clutter, we omit the *pac* prefix from concept and property identifiers.

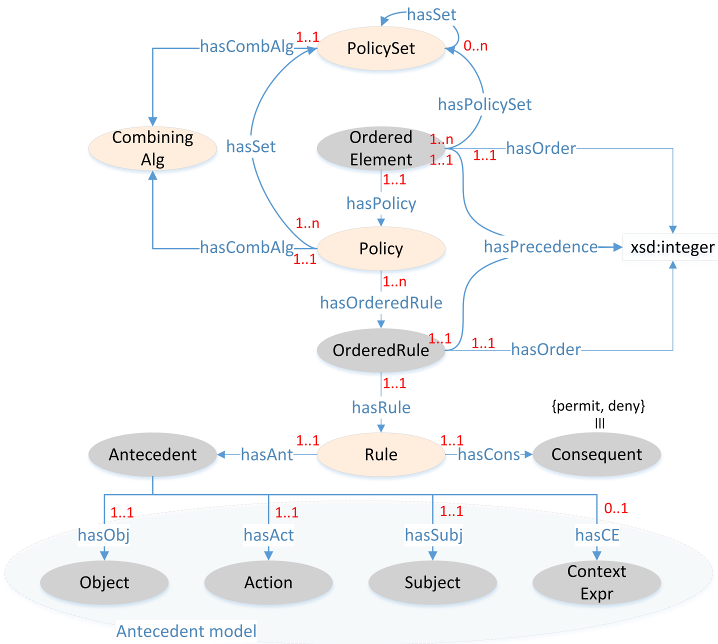

Figure 2: ABAC model (numerical annotations near arrows indicate cardinality constraints).

tively, ABAC policies, ABAC rules, ABAC policy sets and combining algorithms (both rule- and policy-based ones). As depicted in Figure 2, each ABAC policy and policy set is associated, via the property *hasCombAlg*, with *exactly one* combining algorithm, and each ABAC rule is associated, via the properties *hasAnt* and *hasCons*, with *exactly one* antecedent and *exactly one* consequent respectively; the consequent invariably evaluates to one of the individuals[3] *permit* or *deny*.

Our ontological representation also includes the concepts *OrderedRule* and *OrderedElement* (see Figure 2). These are intended to capture the particular *order* that the creator of a policy, or of a policy set, imposes upon the rules of a policy, or upon the elements of a policy set; they are also intended to capture the *precedence* that these rules, or elements, enjoy during their enforcement upon an access request[4]. More specifically, as depicted in Figure 2, each instance of the concept *OrderedRule* is associated, through the property *hasRule*, with *exactly one* ABAC rule and, through each of the properties *hasOrder* and *hasPrecedence*, with *exactly one* non-negative integer that indicates that rule's order and precedence respectively in a containing policy. The *OrderedElement* concept comprises two kinds of element: ordered policies and ordered policy sets. An ordered policy is associated, through the property *hasPolicy*, with *exactly one* ABAC policy; it is also associated — with the same properties as above— with two non-negative integers that represent the policy's order and precedence. Similarly, an ordered policy set is associated, through the property *hasPolicySet*, with *exactly one* (nested) ABAC policy set and, through the same properties as above, with two non-negative integers that represent that policy set's order and precedence. An ABAC policy is associated with its constituent ordered rules through the property *hasOrderedRule* (see Figure 2), whilst an ordered ABAC policy, or an ordered policy set, is associated with its encompassing policy set(s) through the property *hasSet*.

The ontological model outlined above is formally articulated in terms of *terminological* (TBox) and *assertional* (ABox) axioms expressed in the $\mathcal{SROIQ}$ Description Logic[5] (DL) (Horrocks et al., 2006); the interested reader is referred to (Veloudis and Paraskakis, 2015b) for a relevant account.

[3] The terms 'instance' and 'individual' are used interchangeably in this work.

[4] As discussed in Section 5.2, the order that the creator of a policy, or of a policy set, imposes upon the rules of a policy, or the elements of a policy set, does not necessarily coincide with their *enforcement order*: the latter is derived from the precedence values specified by the combining algorithm that is associated with a policy or a policy set.

[5] $\mathcal{SROIQ}$ is the DL underlying OWL 2; in our work, we resort to $\mathcal{SROIQ}$ due to the conciseness and rigorousness of its notation.

3.2.1 Delving into the Antecedent of an ABAC rule

As already stated, the antecedent of an ABAC rule embodies a set of *attributes* whose values need to be taken into account when deciding whether to permit, or deny, a particular access request. Below we elaborate on a set of constraints regarding these attributes. The first constraint states that the antecedent of an ABAC rule *must* embody *exactly one* protected asset. Ontologically, this is captured by demanding that each instance of the concept *Antecedent* is associated with *exactly one* individual from the class *Object* of the CM, and that this association should be realised through the object property *hasObj* (see Figure 2). The second constraint articulates that each ABAC rule must be associated with *exactly one* action from the class *Action* (i.e. with exactly one action that is to be performed on the protected asset), and that this association should be realised through the property *hasAct* (see Figure 2). The third constraint states that each ABAC rule must be associated with *at least one* subject (either human or machine) from the class *Subject* (i.e. with at least one entity requesting access to the protected asset), and that this association should be realised through the property *hasSubj*. Finally, the fourth constraint demands that each ABAC rule refers to *at most one context expression* —i.e. to at most one expression that constrains the values of the contextual attributes that pertain to an access request— and that this association should be realised through the property *hasCE*. Context expressions are further elaborated below.

A context expression (CE) is a propositional logic formula that articulates the *contextual conditions* that must hold in order to permit, or deny, a request. Such conditions may refer to the subject and/or object of a request, or to the request itself. Ontologically, a CE is represented as an instance of the class *ContextExpr* (see Figure 2). The various attributes that it binds, i.e. its *parameters*, are represented as instances of the classes encompassed by the CM (see Figure 1). These parameters are associated with their encompassing CE through the object property *hasParam* and may be combined through the usual propositional logic connectives. As depicted in Figure 3, a CE invariably enjoys *at least one* association with a parameter through the property *hasParam*. Moreover, a CE may be defined recursively, in terms of one or more other CEs; this is captured by including the class *ContextExpr* in both the domain and the range of the property *hasParam*. A CE is attached to the entity that it refers to through the object property *refersTo*. As an example, consider a CE identified by the individual e that articulates that the subject s resides either in the physical location identified as 'EU', or in the network location identified by the subnet 144.0.0.0/8. Ontologically, such a CE is expressed by *asserting* that e is an instance of the abstract class that comprises all those individuals that feature some association through the property *hasParam* with the individual *EU*, or some association through the same property with the individual 144.0.0.0/8, and also some association through the property *refersTo* with the individual s. Such an assertion is formally expressed in $\mathcal{SROIQ}$ as follows[6]:

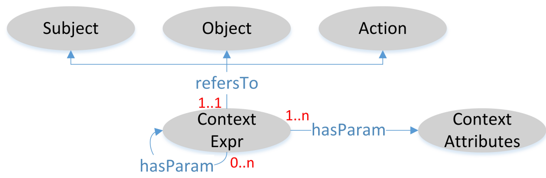

Figure 3: Context expression model.

$$\{e\} \sqsubseteq ((\leq 1\ hasParam.\{EU\}) \sqcup (\leq 1\ hasParam.\{144.0.0.0/8\})) \sqcap (\leq 1\ refersTo.\{s\}) \quad (1)$$

3.3 Access Requests

Similar to the antecedent of an ABAC rule, an access request may embody attributes that specify the particular protected asset that it targets (i.e. its object), the action that it attempts to perform on that object, its subject, as well as a context expression that reflects the relevant contextual circumstances under which it is issued. Ontologically, an access request is represented as an instance of a concept named *Request*; it is associated with its attributes through the same properties as the ones used for associating ABAC rule antecedents with their attributes, namely: *hasObj*, *hasAct*, *hasSubj* and *hasCE*. Note that, in contrast to the case of ABAC rule antecedents, we avoid articulating here any constraints regarding the attributes of an access request. This is because access requests, as opposed to ABAC rules, are externally-generated artefacts whose structure cannot be controlled or predefined. Of course, if a request does not present the structure prescribed by one or more ABAC rules, e.g. if a request fails to convey certain required attributes, it will not trigger the enforcement of these rules.

[6] For any object property P and concept C, $(\leq 1P.C)$ represents the abstract class that comprises all those individuals that have at least one association through P with an instance of C. The symbols $\sqcup$ and $\sqcap$ represent, respectively, class union and intersection.

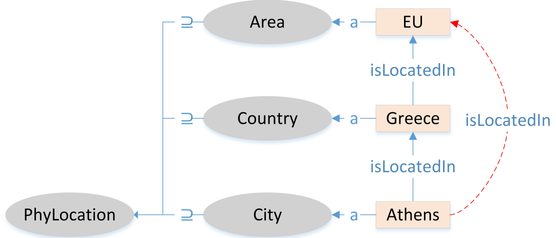

Figure 4: Inferencing at the CM – the perforated line indicates an inferred fact.

4 DETERMINING ENFORCEABILITY FOR ABAC RULES

Reasoning about the enforceability of an ABAC rule upon an access request is a crucial part of our mechanism for determining inter-policy relations. This section outlines how such reasoning is performed.

An ABAC rule r is *enforceable* upon an access request q iff q *entails* r's antecedent, i.e. iff r's antecedent is *semantically inferable* from q. As an example, let r's antecedent enjoy exactly one association, via the properties $hasObj$, $hasAct$, $hasSubj$ and $hasCE$, with the individuals o, w, s and e respectively. e represents the CE of assertion 1 of Section 3.2.1. Suppose now that the request q is associated, via the same properties, with the individuals o, w, s and e' respectively. e' represents a CE that conveys the fact that s resides in, say, the city of Athens. r is enforceable on q as: (i) q is associated with the individuals o, w, s as demanded by r; (ii) e' entails e — an entailment that is based upon semantic inferencing that generates the knowledge that the city of Athens is indeed located in the EU.

Evidently, determining the enforceability of a rule on an access request may entail semantic inferencing in the CM. For such inferencing to be possible, however, certain knowledge artefacts must be encoded in the CM. For instance, in the example above, the concept $PhyLocation$ needs to include such concepts as $Area$, $Country$ and $City$, along with the appropriate instances; it also needs to include the transitive property $isLocatedIn$ that interrelates these instances[7] (see Figure 4).

We are now ready to outline our approach for determining the enforceability of a rule r upon an access request q. Firstly, we programmatically construct an *abstract* OWL class that comprises all those individuals that could, potentially, play the role of r's antecedent. Such an abstract class, call it A_r, includes all those individuals that are associated, through the appropriate properties, with certain *attribute instances* from the CM. For instance, going back to the example above, A_r takes the form (in $\mathcal{SROIQ}$):

$$A_r \equiv ((=1\ hasObj.\{o\}) \sqcap (=1\ hasSubj.\{s\}) \sqcap (=1\ hasAct.\{w\}) \sqcap (=1\ hasCE.C) \quad (2)$$

where[8] C is the abstract class of assertion 1 of Section 3.2. Secondly, the same process is followed for constructing an abstract class, call it R_q, that comprises all those individuals that may potentially play the role of the request q.

The rule r is enforceable upon q iff R_q is a subclass of A_r, i.e. iff $R_q \sqsubseteq A_r$. The validity of this subclass relation is automatically assessed through the use of the Pellet reasoner (Sirin et al., 2007) (although any other DL reasoner could have been used instead).

5 RULE, POLICY AND POLICY SET SUBSUMPTION

We are now ready to present our approach to determining subsumption between ABAC rules, policies and policy sets. Such a determination is based on the access requests that these rules, policies and policy sets may permit or deny. We start off with ABAC rules and then proceed to policies and policy sets.

5.1 Rule Subsumption

For any two ABAC rules r_1 and r_2, r_1 is *subsumed* by r_2 iff r_1 and r_2 are associated with the same decision and r_2 is enforceable on any access request on which r_1 is enforceable. Clearly, for this to be possible, the abstract class of all individuals that could potentially play the role of the antecedent of r_1 must be a subclass of the corresponding class of all individuals that could potentially play the role of the antecedent of r_2, i.e. $A_{r_1} \sqsubseteq A_{r_2}$, where A_{r_1} and A_{r_2} are constructed as outlined in Section 4. For instance, consider the rule r of the example of Section 4 and let r' be another rule whose antecedent enjoys —through the properties $hasObj$, $hasAct$ and $hasSubj$— exactly the same associations as r's antecedent (i.e. with the individuals o, w and s respectively); also, let the antecedent of

[7] These concepts, instances and properties are introduced during the process of priming the CM — a process undertaken by the stakeholders who adopt our framework and aims at tailoring the CM to suit their particular purposes.

[8] For any object property P and concept C, $(=1P.C)$ represents the abstract class that comprises all those individuals that feature *exactly one* association through P with an instance of C; it is an abbreviation for the $\mathcal{SROIQ}$ notation $(\leq 1\ P.C) \sqcap (\geq 1\ P.C)$.

r' be associated, through the property *hasCE*, with an individual e'' that represents a CE that demands that s resides in, say, Greece — formally:

$$\{e''\} \sqsubseteq (\leq 1\ hasParam.\{Greece\}) \sqcap (\leq 1\ refersTo.\{s\}) \tag{3}$$

As long as r and r' are associated with the same decision, r subsumes r' for the former is enforceable on any request that the latter is: any request that originates from Greece (as demanded by the latter) also originates from the EU *or* the subnet 144.0.0.0/8 (as demanded by the former). In other words, semantic inferencing (see Figure 4) allows us to conclude that the abstract class appearing in assertion 3 above is a subclass of the abstract class appearing in assertion 1.

Rule subsumption may be automatically determined by a DL reasoner.

5.2 Policy Subsumption

Determining whether one ABAC policy is subsumed by another requires consideration of the *precedence* values possessed by the rules of these policies; these values are assigned on the basis of the *rule-combining algorithm* associated with each of the policies. For any two rules r_1 and r_2 of a policy p, r_2 takes precedence over r_1, denoted $r_1 <_p r_2$, iff the precedence value associated with r_1 in p (see Section 3.2) is lesser than the precedence value associated with r_2 in p. Let us now briefly outline how precedence values are assigned to rules on the basis of rule-combining algorithms.

5.2.1 Rule-Combining Algorithms

In (XAC, 2013), the following rule-combining algorithms are defined: '*deny overrides*' (DO), '*permit overrides*' (PO) and '*first applicable*' (FA). The DO algorithm imposes the following precedence on p's ruleset S_p: (i) all 'deny' rules (if any) are assigned an equal precedence; (ii) all 'permit' rules (if any) are assigned an equal precedence; (iii) any 'deny' rule is assigned higher precedence than any 'permit' rule. Entirely symmetrical precedence values are imposed by the PO algorithm: (i) all 'permit' rules (if any) are assigned an equal precedence; (ii) all 'deny' rules (if any) are assigned an equal precedence; (iii) any 'permit' rule is assigned higher precedence than any 'deny' rule. Finally, the case of the FA algorithm is quite different for it does not impose by itself any precedence on S_p: instead, it relies on the *order* imposed by p's creator through the *hasOrder* property (see Section 3.2)[9].

5.2.2 Propagating Access Control Decisions to the Policy Level

We now determine how access control decisions are propagated from the rule level to the policy level. For an access request q to invoke the decision v from p (where $v ::= permit|deny$), the following conditions must conjunctively hold. Firstly, there must exist a non-empty subset of S_p, say $S_{p,v}$, that solely comprises rules that yield the decision v, formally: $S_{p,v} = \{r \in S_p | C_r \equiv \{v\}\}$ where C_r is the class that represents r's consequent. Secondly, *at least one* of these rules must be *enforceable* on q. In other words, the abstract class R_q must be a subclass of the union of all the abstract classes A_r where $r \in S_{p,v}$; formally, $R_q \sqsubseteq \bigsqcup_{r \in S_{p,v}} A_r$. Thirdly, there must exist a rule r in $S_{p,v}$ that is enforceable on q such that no other rule (if any) that yields a decision different than v and has a precedence higher than r is enforceable on q. In other words, none of the rules that belong to the set $H_{p,v,r} = \{r' \in S_p | C_r \equiv \{\bar{v}\} \wedge r <_p r'\}$ must be enforceable on q for otherwise one of these rules would be enforced on q yielding a decision different than v ($\bar{v}$ represents here the opposite decision than v). It follows that the class of all requests for which p yields the decision v, denoted $R_{p,v}$, is defined by the following assertion:

$$R_{p,v} \equiv \bigsqcup_{r \in S_{p,v}} (A_r \sqcap \neg(\bigsqcup_{r' \in H_{p,v,r}} A_{r'})) \tag{4}$$

Note that, for any abstract class C, the notation $\neg C$ denotes the complement of C, i.e. the class of all individuals that are not instances of C.

[9] In (XAC, 2013), ordered versions of the DO and PO algorithms are also reported. Nevertheless, the ordered versions of these algorithms drive a policy to respond to an access request with exactly the same decision as their unordered counterparts. It follows that the ordered and unordered versions of the DO and PO algorithms are indistinguishable when determining inter-policy subsumption. This is because, as already discussed, such subsumption is determined on the basis of the manner in which access requests are responded to by a policy. The ordered versions of the DO and PO algorithms will therefore not further concern us here. In addition, the following variations of the DO and PO algorithms are often encountered in practice: '*permit unless deny*' and '*deny unless permit*'. The former algorithm (meaning that p yields by default a 'permit' decision unless one of its 'deny' rules, if any, is enforced) becomes equivalent to the DO algorithm through the inclusion in S_p of a special 'permit' rule that has OWL's universal class as its antecedent (and thus is enforceable upon any access request); an entirely symmetrical account applies to the latter algorithm.

5.2.3 Determining Policy Subsumption

A policy p_1 is considered to be subsumed by a policy p_2 with respect to the decision v, iff the class of all requests for which p_1 yields the decision v is subsumed by the class of all requests for which p_2 yields v, i.e. iff $R_{p_1,v} \sqsubseteq_v R_{p_2,v}$; p_1 is considered to be subsumed by a policy p_2, denoted $R_{p_1} \sqsubseteq R_{p_2}$, iff p_1 is subsumed by p_2 for both values of v. Policy subsumption may be automatically determined by a DL reasoner.

5.2.4 Redundant Rules

In addition to reasoning about policy subsumption, our mechanism may also detect any *redundant* rules in S_p. A rule r is redundant iff it is never enforced on any access request. This occurs when the antecedent of at least one rule from S_p that has higher precedence than r subsumes the antecedent of r; formally: $A_r \sqsubseteq \bigsqcup_{\{r' \in S_{p,v} | r <_p r'\}} A_{r'}$.

5.3 Policy Set Subsumption

We turn now our attention to defining subsumption between ABAC *policy sets*. Recall from Section 3.2 that a policy set may comprise both policies and/or nested policy sets. Let $S_{ps} = \{e_i | i \in [1..n]\}$ be the elements of a policy set *ps*. Determining subsumption between policy sets entails consideration of the *precedence* values possessed by the elements of these sets, hence of the *policy-combining* algorithms associated with them. For any two elements $e_1, e_2 \in S_{ps}$, e_2 takes precedence over e_1 in *ps*, denoted $e_1 <_{ps} e_2$, iff the precedence value associated with e_1 in *ps* is lesser than the precedence value associated with e_2. Let us briefly outline how precedence values are assigned to elements of policy sets on the basis of policy-combining algorithms.

5.3.1 Policy-Combining Algorithms

In (XAC, 2013), the following policy-combining algorithms are defined: '*deny overrides*', '*permit overrides*', '*first applicable*' and '*only one applicable*' (OOA). The DO and PO algorithms, as opposed to their rule-based counterparts, do *not* assign any particular precedence values on the elements of a policy set. The reason for this is as follows. Unlike a policy rule which is inextricably tied to a 'permit' or a 'deny' decision, an element of a policy set may yield *either* decision depending on the access request that it is being applied to; in other words, the same element may yield a 'deny' decision for certain requests and a 'permit' decision for certain others. It follows that no precedence value can be assigned to the elements of a policy set on the basis of the decisions that they entail. Turning now to the FA algorithm, this is similar to its rule-based counterpart: the precedence values assigned to the elements of a policy set coincide with the *order* of these elements as this has been defined by the creator of the policy set. Finally, the OOA algorithm ensures that exactly one element of a policy set is applicable to a request and responds to that request with that element's decision; if two or more elements are applicable, an '*Indeterminate*' decision is returned, whereas if no elements are applicable, the decision is '*NotApplicable*'. As we would expect, the OOA algorithm does not impose any precedence to the elements of a policy set.

5.3.2 Propagating Access Control Decisions to the Policy Set Level

We now determine how access control decisions are propagated to the policy set level. Let $R^{CA}_{ps,v}$ denote the class of all requests that invoke the decision v from a policy set *ps*, where CA::=$DO|PO|FA|OOA$. The fact that some combining algorithms do not assign precedence values to policy set elements precludes the provision of a single assertion, analogous to assertion 4 of Section 5.2.2, that characterises all requests that may be permitted, or denied, by a policy set without taking into account the combining algorithm associated with that policy set. We thus consider each policy-combining algorithm in turn; initially we focus on policy sets that only comprise policies.

Deny Overrides and Permit Overrides We start with the DO algorithm. The class of all access requests for which *ps* yields a 'deny' decision is defined as:

$$R^{DO}_{ps,deny} \equiv \bigsqcup_{e \in S_{ps}} R_{e,deny} \quad (5)$$

These are effectively all requests which cause at least one element of S_{ps} to yield a 'deny' decision. Similarly, the class of all requests for which *ps* yields a 'permit' decision is defined as:

$$R^{DO}_{ps,permit} \equiv (\bigsqcup_{e \in S_{ps}} R_{e,permit}) \sqcap \neg R^{DO}_{ps,deny} \quad (6)$$

These are effectively all requests which cause one or more elements of S_{ps} to yield a 'permit', but not a 'deny', decision.

The case of the PO algorithm is entirely analogous. The class of all access requests for which *ps* yields a 'permit' decision is defined as:

$$R^{PO}_{ps,permit} \equiv \bigsqcup_{e \in S_{ps}} R_{e,permit} \quad (7)$$

Similarly, the class of all requests for which *ps* yields a 'deny' decision is defined as:

$$R^{PO}_{ps,deny} \equiv (\bigsqcup_{e \in S_{ps}} R_{e,deny}) \sqcap \neg R^{PO}_{ps,permit} \quad (8)$$

First Applicable The class of all access requests for which *ps* yields *v* is defined as:

$$R^{FA}_{ps,v} \equiv \bigsqcup_{e \in S_{ps}} (R_{e,v} \sqcap \bigsqcup_{e' \in S_{ps} \wedge e <_{ps} e'} R_{e',\bar{v}}) \quad (9)$$

These are effectively all requests which cause an element e in S_{ps} to yield v without causing any element e' in *ps* with a precedence higher than e to yield a decision different than v (denoted as $\bar{v}$).

Only One Applicable The class of all access requests for which *ps* yields the decision v is defined as:

$$R^{OOA}_{ps,v} \equiv \bigsqcup_{e \in S_{ps}} (R_{e,v} \sqcap \neg \bigsqcup_{e' \in S_{ps} \wedge e \neq e'} (R_{e',deny} \sqcup R_{e',permit})) \quad (10)$$

These are effectively all requests which cause *exactly one* element in S_{ps} to yield v.

The class $R^{CA}_{ps,v}$ may be generalised to policy sets that potentially comprise nested policy sets as follows: the class $R_{e,v}$ in assertions 5-10 above is replaced by a construct $X_{e,v}$ which evaluates to $R_{e,v}$ if e is a policy, and to $R^{CA}_{e,v}$ if e is a policy set where CA is the combining algorithm associated with e.

5.3.3 Determining Policy Set Subsumption

We are now ready to define subsumption between policy sets. A policy set ps_1 is considered to be subsumed by a policy set ps_2 with respect to the decision v, iff the class of all requests for which ps_1 yields the decision v is subsumed by the class of all requests for which ps_2 yields the decision v, i.e. iff $R^{CA_1}_{ps_1,v} \sqsubseteq_v R^{CA_2}_{ps_2,v}$ where CA_1 and CA_2 are the combining algorithms associated with ps_1 and ps_2 respectively. ps_1 is considered to be subsumed by ps_2, denoted $R^{CA_1}_{ps_1} \sqsubseteq R^{CA_2}_{ps_2}$, iff p_1 is subsumed by p_2 for both values of v.

The efficiency with which subsumption relations are checked by our mechanism naturally depends on the complexity of the underlying CM, of the policies and the policy sets, as well as of the CEs attached to policy rules. Although we have not performed any extensive performance analysis, our mechanism has been able to check subsumption between average-complexity policies over a moderate size CM (comprising 1210 OWL axioms) in the order of 2.5 to 3 seconds. It is to be emphasised here that our mechanism is intended to be used during application design time for assisting developers in devising effective policies. In this respect, performance is not a critical feature and moderate variations in the aforementioned latency are tolerable. The use of our framework for evaluating access requests in real-time has been extensively discussed in (Veloudis et al., 2019).

6 RELATED WORK

The work reported in (Kolovski et al., 2007) presents a formalisation of XACML using DLs. Similar to our work, it proposes an approach to reasoning about subsuming policies based on a semantic characterisation of the class of access requests for which these policies emit identical decisions. Nevertheless, as opposed to our work, it fails to provide an underlying ontological model for rules, policies and policy sets. This entails the following disadvantages. Firstly, it fails to explicitly model combining algorithms, hence the precedence that they impose on policies and policy sets. To overcome this problem, the work in (Kolovski et al., 2007) employs a formalism alien to OWL, namely Defeasible DLs (DDLs), for capturing precedence implicitly. Nevertheless, such an approach incurs the overhead of reducing provability in DDLs to concept satisfiability in OWL; in addition, as the authors concede, it hinders the expression of certain combining algorithms such as the OOA algorithm. Secondly, it fails to model the potentially complex contextual knowledge that may be associated with rules and thus precludes the performance of semantic inferencing for detecting subsumption between rule *attributes*, in particular between CEs. This is clearly a drawback in dynamic cloud environments.

On a different note, a number of approaches have been proposed for semantically representing policies (Uszok et al., 2004; Kagal et al., 2003; Nejdl et al., 2005). These generally rely on OWL (OWL, 2004) for capturing the various knowledge artefacts that reside in policies. In (Uszok et al., 2004) KaoS is presented — a generic framework offering: (i) a human interface layer for the expression of policies; (ii) a policy management layer that is capable of resolving conflicting policies; (iii) a monitoring and enforcement layer that encodes policies in a programmatic format suitable for enforcing them. KaoS lacks any mechanism for explicitly reasoning about subsuming relations between access control policies.

In (Kagal et al., 2003) Rei is proposed: a framework for specifying, analyzing and reasoning about policies. Similar to our work, a policy comprises a list of rules that take the form of OWL properties; it also comprises a context that defines the underlying policy domain. Rei resorts to the use of constructs adopted from rule-based programming languages for the definition of policy rules. This essentially prevents Rei from exploiting the full inferencing potential of OWL as policy rules are expressed in a formalism external to OWL. In addition, it does not provide any mechanism for reasoning about subsumption in access control policies.

In (Nejdl et al., 2005) the authors propose POLICYTAB for facilitating trust negotiation in Semantic Web environments. POLICYTAB adopts ontologies for the representation of policies that guide a trust negotiation process ultimately aiming at granting, or denying, access to sensitive Web resources. These policies essentially specify the credentials that an entity must possess in order to carry out an action on a sensitive resource that is under the ownership of another entity. Nevertheless, no attempt is made to semantically model the context associated with access requests, rendering POLICYTAB inadequate for the dynamic nature of cloud environments.

Finally, markup languages (e.g. (Boley et al., 2016; SAM, 2008; XAC, 2013)) provide declarative formalisms for the specification of policies. Nevertheless, they do not provide any means of capturing the knowledge encoded in policies hence reaching semantic agreement beyond the boundaries of the organisations that adopt them. This can only lead to ad-hoc reasoning about inter-policy relations.

7 CONCLUSIONS

We have presented an approach to reasoning about subsumption between access control policies in dynamic cloud environments. The reasoning is based on a semantic characterisation of all those access requests to which the policies respond in an identical manner, whilst it is performed automatically through semantic inferencing that is carried out by off-the-shelf reasoners.

As part of future work we intend to perform further performance tests in order to more accurately determine the scalability of our approach to larger underlying CMs. In addition, we intend to incorporate our approach in an editor that we are currently developing for facilitating the construction of ABAC rules, policies and policy sets. This way, each time a new rule, policy or policy set is created, the editor will determine whether it is subsumed by an existing rule, policy or policy set and thus assist developers in devising *effective* access control policies and policy sets.

REFERENCES

(2004). *W3C Recommendation. 2004. OWL Web Ontology Language Reference*. W3C. `https://www.w3.org/TR/owl-ref`.

(2008). *Security Assertions Markup Language (SAML) Version 2.0. Technical Overview*. OASIS. `https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf`.

(2013). *eXtensible Access Control Markup Language (XACML) Version 3.0*. OASIS. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`.

(2015). Paasword - a holistic data privacy and security by design platform-as-a service framework. `https://www.paasword.eu`.

(2015). *What's Hindering the Adoption of Cloud Computing in Europe?* Cloud Security Alliance. `https://blog.cloudsecurityalliance.org/2015/09/15/whats-hindering-the-adoption-of-cloud-computing-in-europe/`.

Boley, H., Athan, T., Paschke, A., Giurca, A., Bassiliades, N., Governatori, G., Palmirani, M., Wyner, A., and Zou, G. (2016). Specification of deliberation ruleml 1.01. `http://wiki.ruleml.org/index.php/Specification\_of\_Deliberation\_Rule\linebreak[0]ML\_1.01`.

Horrocks, I., Kutz, O., and Sattler, U. (2006). The even more irresistible sroiq. In Doherty, P., Mylopoulos, J., and Welty, C. A., editors, *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 57–67. AAAI Press.

Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., and Scarfone, K. (2014). *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. NIST. `http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-162.pdf`.

Kagal, L., Finin, T., and Joshi, A. (2003). A policy language for a pervasive computing environment. In *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, pages 63–74.

Kolovski, V., Hendler, J., and Parsia, B. (2007). Analyzing web access control policies. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 677–686, New York, NY, USA. ACM.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., and Leaf, D. (2011). *NIST Cloud Computing Reference Architecture*.

Nejdl, W., Olmedilla, D., Winslett, M., and Zhang, C. C. (2005). Ontology-based policy specification and management. pages 290–302, Berlin, Heidelberg. Springer Berlin Heidelberg.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semant.*, 5(2):51–53.

Uszok, A., Bradshaw, J., Jeffers, R., Johnson, M., Tate, A., Dalton, J., and Aitken, S. (2004). KAoS policy management for semantic web services. *IEEE Intel. Sys.*, 19(4):32–41.

Veloudis, S. and Paraskakis, I. (2015a). *Access Policies Model*. PaaSword Project Deliverable D2.2.

Veloudis, S. and Paraskakis, I. (2015b). *Access Policies Model*. PaaSword Project Deliverable D3.2.

Veloudis, S. and Paraskakis, I. (2016). Defining an ontological framework for modelling policies in cloud environments. In *8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom'16)*.

Veloudis, S., Paraskakis, I., and Petsos, C. (2017a). An ontological framework for determining the repercussions of retirement actions targeted at complex access control policies in cloud environments. In *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing*. ACM.

Veloudis, S., Paraskakis, I., Petsos, C., Verginadis, Y., Patiniotakis, I., Gouvas, P., and Mentzas, G. (2019). Achieving security-by-design through ontology-driven attribute-based access control in cloud environments. *Future Generation Computer Systems*, 93:373–391.

Veloudis, S., Paraskakis, I., Petsos, C., Verginadis, Y., Patiniotakis, I., and Mentzas, G. (2017b). An ontological template for context expressions in attribute-based access control policies. In *Proceedings of the 7th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,*, pages 151–162. INSTICC, ScitePress.

Veloudis, S., Verginadis, Y., Paraskakis, I., Patiniotakis, I., and Mentzas, G. (2016). Context-aware security models for paas-enabled access control. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016) Vol. 1 and 2*, pages 201–212. INSTICC, ScitePress.

Verginadis, Y., Patiniotakis, I., and Mentzas, G. (2015). *Context-aware Security Model, PaaSword Project Deliverable D2.1*. `https://www.paasword.eu/wp-content/uploads/2016/09/D2-1\_Context-awareSecurityModel.pdf`.