Foundations for Designing, Defining, Validating and Executing 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
{sveloudis, iparaskakis, chpetsos}@seerc.org

Abstract. By embracing cloud computing enterprises are able to boost their agility and productivity whilst realising significant cost savings. However, due to security and privacy concerns, many enterprises are reluctant to migrate their data and operations to the cloud. One way to alleviate these concerns is to devise access control policies that infuse suitable security controls into cloud services. Nevertheless, the complexity inherent in such policies, stemming from the dynamic nature of cloud environments, calls for a framework that provides *assurances* with respect to the *effectiveness* of the policies. In this respect, this work proposes a class of constraints, the so-called *well-formedness constraints*, that provide such assurances by empowering stakeholders to harness the *attributes* of the policies. Both the policies and the constraints are expressed *ontologically* hence enabling automated reasoning about the abidance of the policies with the constraints.

Keywords: Foundation framework for policies · Designing policies · Defining policies policy governance · Access control · Policy governance · Ontologies · Description logics

1 Introduction

Cloud computing enables enterprises to realise significant cost savings, whilst boosting their agility and productivity. Nevertheless, due to security and privacy concerns, many enterprises are reluctant to relinquish control of—oftentimes critical—corporate assets by migrating their data and applications to third-party cloud providers [1]. One way to alleviate these concerns, hence bolster the adoption of cloud computing, is to infuse adequate *access control policies* into the applications through which critical assets are accessed in the cloud [2]. Nevertheless, the inherently dynamic nature of cloud environments calls for policies that are able to incorporate a potentially complex body of *contextual knowledge* pertaining to access requests [3]. As an example, consider a policy whereby a particular entity (s) is allowed to read a sensitive data object (o) only when: (i) o resides in a data centre in the EU; (ii) s resides in a specific geographical area (say the city of Athens), or the request originates from a particular subnet; (iii) the request is received during a prescribed time interval.

F. De Paoli et al. (Eds.): ESOCC 2017, LNCS 10465, pp. 75–82, 2017.
DOI: 10.1007/978-3-319-67262-5_6

We argue that, for stakeholders to entrust such complex access control policies with the protection of their sensitive assets, a framework that provides assurances about the *effectiveness* of the policies is required [2]. In particular, a framework is required that assists developers in infusing effective access control policies into the applications through which sensitive assets are accessed in the cloud. Our work, conducted as part of the PaaSword project [4], provides such a framework. More specifically, it offers a generic *security-by-design* solution—essentially a PaaS offering—that provides assurances about the effectiveness of *context-aware* access control policies by facilitating their *governance*. To this end, it draws upon a *semantic representation* of policies, one that *ontologically* captures the various knowledge artefacts that are encoded in the policies. Such a representation disentangles the expression of policies from the actual code of the applications into which they are infused hence enabling automated reasoning about their *correctness*.

This paper proposes an approach to such reasoning. In particular, it proposes a set of *ontologically-expressed constraints*, the so-called *well-formedness constraints*, that articulate all those *knowledge artefacts* that *must*, *may* or *must not* be embodied in an access control policy. These constraints give rise to a *higher-level ontology*, one that specifies an allowable *form*, or *structure*, by which access control policies must abide. Evidently, well-formedness constraints empower stakeholders to harness the knowledge artefacts embodied in access control policies that protect their sensitive assets. In other words, they empower stakeholders to infuse into these policies their business logic and overall stance towards security. In this respect, well-formedness constraints assist developers in devising policies that are appropriate for the stakeholders' needs, hence for the assets that they protect.

The rest of this paper is structured as follows. Section 2 presents an ontological representation for access control policies and well-formedness constrains. Section 3 outlines a mechanism that reasons about the satisfaction of well-formedness constraints. Section 4 discusses related work and Sect. 5 outlines conclusions.

2 Constraining Access Control Policies

As already discussed, 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 [5], due to their inherent generality stemming from their inherent reliance on the generic concept of an *attribute*, are particularly suitable for capturing such knowledge [3] and are thus adopted in our work. This section outlines an OWL-based representation for ABAC policies and *well-formedness constraints*; as already mentioned, the latter harness the attributes embodied in the former.

2.1 A Model for ABAC Rules and Policies

Following the XACML standard [6], an ABAC policy comprises one or more ABAC *rules*. Upon receipt of an access request, a rule-combining algorithm [6] is executed in order to select which one of these rules, if any, will be applied in order to arrive at a ‘permit’ or a ‘deny’ decision. 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 is considered ‘Not Applicable’ or ‘Indeterminate’ [6]).

An ABAC rule comprises an *antecedent* and a *consequent*. The latter specifies the rule’s *decision*, which according to the XACML standard, invariably resolves to either a ‘*permit*’ or a ‘*deny*’. 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 *attributes*, whose values need to be taken into account when deciding whether to permit, or deny, a request. These attributes are drawn from an underlying Context Model (CM)—an extensible ontological framework that includes interrelated concepts suitable for capturing attributes and the properties thereof. A simplified view of the CM that is used in this work, one which includes only concepts and properties considered in this paper, is depicted in Fig. 1 (for more details on the CM, the interested reader is referred to [7]).

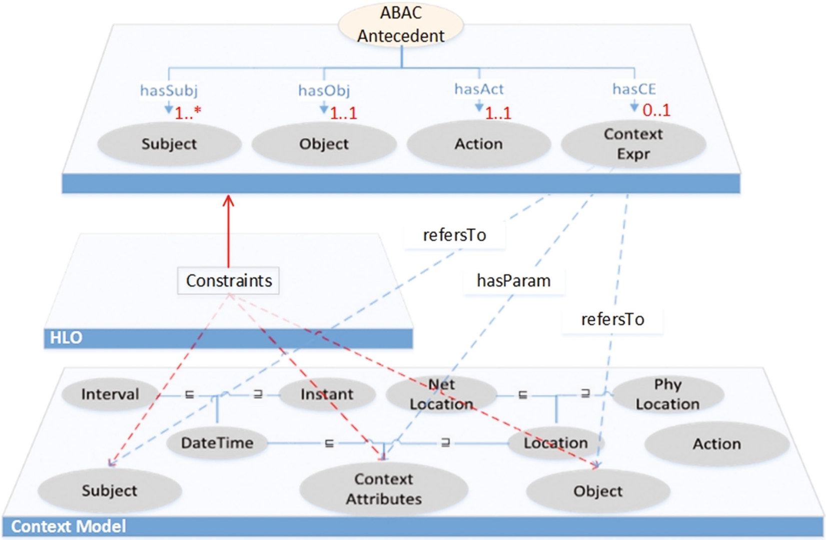

Fig. 1. HLO constraints

Ontologically, ABAC policies are represented as instances of the concept *ABACPolicy*, and ABAC rules as instances of the concept *ABACRule*; ABAC policies are associated with their constituent rules through the object property *hasABACRule*. The antecedent and consequent of an ABAC rule are represented, respectively, as instances of the concepts *ABACAnt* and *ABACCons*; an ABAC rule is associated with its antecedent and consequent via the properties *hasABACAnt* and *hasABACCons* respectively. In addition, the following *restrictions* apply. Firstly, an ABAC policy is invariably associated with *at least one* ABAC rule; secondly, an ABAC rule is invariably associated with *exactly one* antecedent and *exactly one* consequent; thirdly, the consequent of an ABAC rule always resolves to either a ‘permit’ or a ‘deny’ decision (represented respectively by the individuals *permit* and *deny*). All three restrictions are ontologically captured in terms of *terminological* (*TBox*) *axioms* expressed in the $\mathcal{SROIQ}$ Description Logic (DL) [8]. These axioms are presented in Table 1. The first demands that each ABAC policy, i.e. each instance of the concept *ABACPolicy*, is also an instance of the (abstract) class that comprises all those individuals that have *at least one* association through the property *hasABACRule* with an individual from the concept *ABACRule*. The second demands that each ABAC rule, i.e. each instance of *ABACRule*, is also an instance of the class that comprises all those individuals that have *exactly one* association through each of the properties *hasABACAnt* and *hasABACCons* with individuals from the concepts *ABACAnt* and *ABACCons* respectively. Finally, the third axiom demands that the class *ABACCons* comprises solely the individuals *permit* and *deny*.

Table 1. ABAC policy model restriction axioms

Axiom 1	$ABACPolicy \sqsubseteq\ \leq 1 hasABACRule.ABACRule$
Axiom 2	$ABACRule \sqsubseteq (\geq R_i.C_i) \sqcap (\leq 1 R_i.C_i)$ where $i = 1, 2$ and $R_i \equiv hasABACAnt, C_i \equiv ABACAnt$, for $i = 1$ $R_i \equiv hasABACCons, C_i \equiv ABACCons$, otherwise
Axiom 3	$ABACConsequent \equiv \{permit, deny\}$

2.2 Well-Formedness Constraints

Well-formedness constraints specify the *attributes* of an ABAC rule, i.e. all those *knowledge artefacts* from the underlying CM that *must*, *may* or *must not* be embodied in the antecedent of an ABAC rule. In this respect, well-formedness constraints give rise to a *higher-level ontology* (HLO) that defines an allowable *form*, or *structure*, for the antecedent of an ABAC rule (see Fig. 1). The HLO not only articulates the permissible knowledge artefacts embodied in the antecedent, but goes a step further to determine the allowable *cardinalities* with which these artefacts may appear, as well as the allowable *values* that they may assume.

We now briefly elaborate on the HLO constraints that have been devised for ABAC rules in the frame of the PaaSword project. These constraints are ontologically expressed in terms of $\mathcal{SROIQ}$ TBox axioms which restrict the class *ABACAnt*. It is to be noted here that these constraints are *malleable* in the sense that they can be altered to express alternate structures for the antecedent of ABAC rules—i.e. structures that potentially reflect more accurately the application-specific needs of an organisation adopting the PaaSword framework. This malleability is of utmost significance for it empowers stakeholders to infuse into access control policies their business logic and overall stance towards security.

The first constraint states that each ABAC rule *must* embody *exactly one* protected asset. Ontologically, this is captured through a TBox axiom that demands that the antecedent of an ABAC rule, i.e. each instance of the concept *ABACAnt*, 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*. Table 2 provides a formal expression of this axiom, as well as of the rest of the axioms outlined in this section. Similarly, the second axiom states that each ABAC rule must be associated, through the property *hasAct*, with *exactly one* action from the class *Action* (i.e. with exactly one action to be performed on the protected asset); the third axiom states that each ABAC rule must be associated with *at least one* subject from the class *Subject* (i.e. with at least one entity requesting access to the protected asset), and the fourth axiom demands that each ABAC rule *may* refer, via the property *hasCE* 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. Context expressions take the form of instances of the class *ContextExpr* (see Fig. 1) and are further discussed below.

Table 2. HLO axioms

Axiom 1	$ABACAnt \sqsubseteq (\leq 1 hasObj.Object) \sqcap (\geq 1 hasObj.Object)$
Axiom 2	$ABACAnt \sqsubseteq (\leq 1 hasAct.Action) \sqcap (\geq 1 hasAct.Action)$
Axiom 3	$ABACAnt \sqsubseteq \leq 1 hasSubj.Subject$
Axiom 4	$ABACAnt \sqsubseteq \geq 1 hasCE.ContextExpr$

A context expression (CE) is a propositional logic expression that is attached to the antecedent of an ABAC rule and articulates the *contextual conditions* that must hold in order to permit, or deny, a request. These contextual conditions may refer to the subject and/or object of a request, or to the request itself. In other words, a CE captures the body of contextual knowledge that must be taken into account when deciding upon a request. Ontologically, a CE is represented as an instance of the class *ContextExpr* (see Fig. 1). The various attributes that it binds, i.e. its *parameters*, are represented as instances of the CM—in particular, as instances of the classes encompassed by the *ContextAttributes* concept. These parameters are associated with their encompassing CE through the object property *hasParam* and may be combined through the usual propositional logic connectives. A CE invariably enjoys at least one association with a parameter; ontologically, this is captured by an axiom analogous to Axiom 3 of Table 2. 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*. Finally, a context expression is attached to the entity that it refers to through the object property *refersTo*.

The HLO may encompass constraints that restrict the allowable forms that a CE can assume when attached to a particular ABAC rule. These constraints restrict the *cardinalities* with which certain knowledge artefacts from the class *ContextAttributes* may appear in a CE, as well as the allowable *ranges of values* that these artefacts may assume. As an example, consider an HLO constraint that demands that *any* CE attached to an ABAC rule should invariably incorporate at least one parameter that confines the whereabouts of the subject *s* of a request to the physical location identified as *Athens*, or to the network location identified by the subnet 123.0.0.0/8. Ontologically, this constraint takes the form:

$$ContextExpression \sqsubseteq (\leq 1 refersTo.\{s\}) \sqcap ((\leq 1 hasParam.\{Athens\}) \sqcup (\leq 1 hasParam.\{123.0.0.0/8\})) \tag{1}$$

3 Reasoning About the Correctness of ABAC Policies

Reasoning about the correctness of an ABAC rule, hence about the correctness of an ABAC policy that resolves to that rule, involves reasoning about the abidance of the rule by the HLO constraints. Below, we outline how this reasoning is performed by a mechanism that we have developed as part of the PaaSword project. As an example, suppose the following set of $\mathcal{SROIQ}$ *axioms* that articulate the attribute values associated with an ABAC rule; we shall term such an axiom-set a *knowledge base* (KB) [9].

$$\begin{aligned}\mathcal{R} \equiv \{&ABACRule(r), ABACAnt(a), Object(o), Subject(s),\\ &ContextExpr(e), PhyLocation(Athens), hasABACAnt(r,a),\\ &hasABACCons(r, permit), hasObj(a,o), hasSubj(a,s),\\ &hasCE(a,e), hasParam(e, Athens), refersTo(e,s)\}\end{aligned} \tag{2}$$

According to $\mathcal{R}$, the antecedent a of the ABAC rule r is associated with the object o, the subject s and the context expression e; e is further associated with the (physical) location parameter *Athens* which refers to s.

Two seminal assumptions underpinning OWL are the Open-World Assumption (OWA) and the non-Unique Name Assumption (non-UNA). Nevertheless, these assumptions render the use of OWL cumbersome when reasoning about *constraint satisfaction*. Consider, for example, the KB $\mathcal{R}$ above. $\mathcal{R}$ fails to specify the *action* that is to be performed upon the object o. However, according to the OWA, this does not mean that the rule r described by $\mathcal{R}$ does not have such an action associated with its antecedent: it merely means that this association is not specified in $\mathcal{R}$. In order to overcome this obstacle, we adopt the approach proposed in [9] and dispense with the OWA and the non-UNA, effectively enabling *closed-world reasoning* when checking the abidance of ABAC rules by HLO constraints. This reasoning is based on an extended semantics of OWL, namely the Integrity Constraint semantics [9]; an outline of how such reasoning is performed is in order.

Each HLO axiom is translated into a *query*, one that is posed to the KB under validation with the aim of discovering any individuals that *violate* the axiom: if the query returns an empty set of individuals, the axiom is considered to hold; otherwise, it is considered to be violated. The query is, in fact, an assertion axiom that uses variables in place of individuals and expresses the *negation* of the HLO axiom that it translates. As an example, consider Axiom 2 of Table 2. This axiom is translated into a query that attempts to discover in $\mathcal{R}$ any individuals that belong to the class *ABACAnt* and which either enjoy no associations (through the property *hasAct*) with instances of the class

Action, or enjoy two or more such associations with distinct instances of *Action*. Formally:

$$\begin{aligned}&ABACAnt\,(x) \wedge (\mathbf{not}(hasAct(x, y) \wedge Action(y))\vee \\ &(hasAct(x, y) \wedge hasAct(x, z) \wedge Action(y) \wedge Action(z) \wedge \mathbf{not}(y = z))\end{aligned} \tag{3}$$

These queries are termed in [9] *Distinguished Conjunctive Queries* with Negation as Failure (DCQ$^{\mathbf{not}}$). DCQ$^{\mathbf{not}}$ are posed to the KB under validation as SPARQL queries [10]. SPARQL queries are executed in the Pellet reasoner [11] (however, any other OWL reasoner could have been used instead). In [9], a set of translation rules for turning a $\mathcal{SROIQ}$ axiom into a DCQ$^{\mathbf{not}}$, hence into a SPARQL query, is presented.

4 Related Work

A number of approaches have been proposed for the semantic representation of policies [12–14]. These generally rely on OWL [15] for capturing the various knowledge artefacts that underpin the definition of a policy. In [12] 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 automatically checking the correctness, hence the effectiveness, of policies.

In [13] 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 that is alien to OWL. In addition, it does not provide any mechanism for reasoning about the effectiveness of policies.

In [14] 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.

5 Conclusions

We have presented an approach to reasoning about the correctness, hence the effectiveness, of access control policies in dynamic cloud environments. The correctness is judged on the basis of *ontologically-expressed* constraints, the so-called *HLO* *constraints*. The reasoning is based on an extended semantics of OWL, one that dispenses with the OWA and the non-UNA, allowing the transformation of the constraints into queries that are posed to the KBs that represent the rules under validation.

Acknowledgements. The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644814.

References

1. Cloud Security Alliance: What's Hindering the Adoption of Cloud Computing in Europe? Cloud Security Alliance (2015). https://blog.cloudsecurityalliance.org/2015/09/15/whats-hindering-the-adoption-of-cloud-computing-in-europe/. Accessed 6 May 2017
2. Veloudis, S., Paraskakis, I.: Defining an ontological framework for modelling policies in cloud environments. In: CloudCom 2016 – Proceedings of the 8th IEEE International Conference on Cloud Computing Technology and Science, pp. 277—284. IEEE Computer Society, Los Alamitos (2016)
3. Veloudis, S., Paraskakis, I., Petsos, C., Verginadis, Y., Patiniotakis, I., Mentzas, G.: An ontological template for context expressions in attribute-based access control policies. In: CLOSER 2017 – Proceedings of the 7th International Conference on Cloud Computing and Services Science, pp. 123–134. Scitepress (2017)
4. PaaSword project. http://www.paasword.eu/. Accessed 6 May 2017
5. Hu, V.C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller R., Scarfone, K.: Guide to Attribute Based Access Control (ABAC), Definition and Considerations. NIST (2014)
6. eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. OASIS Standard. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html. Accessed 6 May 2017
7. PaaSword Deliverable 2.1. https://www.paasword.eu/deliverables/. Accessed 6 May 2017
8. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible *SROIQ*. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), pp. 57–67. AAAI Press (2006)
9. Tao, J., Sirin, E., Bao, J., McGuinness, D.L.: Integrity constraints in OWL. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10), Atlanta, Georgia, USA, 11–15 July 2010
10. SPARQL 1.1 Query Language W3C Recommendation, 21 March 2013. https://www.w3.org/TR/sparql11-query/. Accessed 6 May 2017
11. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. Web Semant. Sci. Serv. Agents World Wide Web **5**(2), 51–53 (2007)
12. Kagal, L., Finin, T., Joshi, A.: A policy language for a pervasive computing environment. In: Proceedings IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), pp. 63–74. IEEE Computer Society, Washington, D.C. (2003)
13. Nejdl, W., Olmedilla, D., Winslett, M., Zhang, C.C.: Ontology-based policy specification and management. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005, vol. 3532, pp. 290–302. Springer, Heidelberg (2005)
14. Uszok, A., Bradshaw, J., Jeffers, R., Johnson, M., Tate, A., Dalton, J., Aitken, S.: KAoS policy management for semantic web services. IEEE Intel. Syst. **19**(4), 32–41 (2004)
15. OWL 2 Web Ontology Language Primer, 2nd edn. https://www.w3.org/TR/owl2-primer/. Accessed 6 May 2017