Towards Design an Interoperability Framework for Security Policy Languages

Amir Aryanpour¹, Song Y. Yan¹, Scott Davies², Andrew Harmel-law²
¹Department of Computer Science and Technology, University of Bedfordshire, UK
²Technology Services, Capgemini UK Plc

Abstract -- Security policy languages ranged from a single role based access control (RBAC) to a highly sophisticated policy language which is capable of negotiating across a network, all aim to express clearly and concisely what the protection mechanisms are to achieve. However, these policy languages have often been designed independently and hence translation between these languages is difficult. The challenge of representing policies in different languages on a distributed network where there are multiple heterogeneous policy languages in use, affects the main benefit of policy-based security management, namely the enablement of policies and services be controlled and managed at a high level regardless of the adopted underlying policy language.

Through our research we have focused on this inability of transforming and translating policy languages. We intend to provide a framework which will facilitate the representation of security policy languages. However as a first step towards our goal, this paper details the steps which have been taken to theoretically prove that policy languages can be interoperable.

I. INTRODUCTION

The notation of protecting networked resources came to life at the very same moment computer networking was introduced. Many access control model and policy languages have been proposed in order to address the abovementioned concern. These languages, which have undergone a revolution during the last decade, usually come with different specifications, all of which aim to tackle different business requirements. When it comes to distributed networks, which includes virtual environments, different policies possibly written in heterogeneous policy languages and from different domains, could relate to many different resources. On the other hand, resources are often shared amongst different domains. In such a scenario, various policies must be integrated into a single policy in order to govern the access to the jointly owned resources. This has already been investigated and researched by others. Bearing in mind the fact that policy languages are not easily capable of being put into another language, this research however is intended to investigate the management of such an environment and to study the security policies at a more understandable and abstract level.

The lack of an interoperability framework for policy languages has received limited attention on previous occasions. Clemente et al presented a solution for this business requirement in [1]. In addition Basile et al proposed a system to address the problem of policy translation in [2]. Through our research we have re-examined these papers from different points of views. The following facts are applicable to both of them.

The proposed solutions on these papers focus on ontology based policy languages. The extensibility at runtime and adaptability of semantic policy languages convinced the authors of the abovementioned articles to choose semantically rich languages such as OWL over traditional languages. However, brief research shows that only a small set of available policy languages and frameworks which are widely used, are based on semantic languages. Hence, by imposing such a constraint on the framework, a large number of policy languages will not be able to use the solutions. In addition, despite the fact that semantic policy languages usually come with strong underlying formalism, both reports fail to demonstrate the possibility of cross-language policy translations using formal specifications. Undoubtedly we have been inspired by these papers in our research but have aimed to improve their research accordingly.

Figure (1) which serves crucial illustration purposes, displays a typical distributed network which utilises heterogeneous policy languages across different domains. In such an environment the management of domains at an abstract level is challenging if not impossible unless we introduce an abstract management policy framework to control the environment from a single point.

In essence, the policy framework as illustrated below is responsible for providing a platform to policy languages in order to make them interoperable which would undoubtedly be the main contribution of this research. However, before we architect the system and as a prerequisite to our framework’s design, we would like to theoretically prove
that such a translation is possible. The four steps which must be considered on route to this goal can be summarised as:

1. Policy Language Candidates
   Due to the fact that there are too many policy languages to choose from [3], we must classify policy languages into different categories. We must then select a policy language candidate from each individual group which represents the characteristics of that specific set. The result of this step is a set of policy language candidates which will then proceed on to the next step.

2. Algebra Candidate
   Logically, the most appropriate way to translate many-to-many languages is to translate them to and from an abstract language. We choose algebra as the abstract language throughout our PoC (Proof of Concept) hence the research must identify and utilise an appropriate algebra for policy languages.

3. Evaluation of Selected Algebra
   The algebra selected should be capable of expressing different scenarios written in possibly heterogeneous policy languages. Hence it should be evaluated against the policy language candidates chosen in step1.

4. Limitation and Solution
   During the final step, research must identify any existing weakness of the algebra, and formulate a solution to address that accordingly. Although the main contribution of the research is the design and development of the framework itself, but this step of the PoC can be presented as the contribution of this paper.

The rest of this paper is organised into three parts as follows: The first part details a comparison of policy languages and lists a set of policy language candidates. The second part describes the algebra we have used to theoretically prove that the translation of policy languages is achievable. Finally, the last part describes the limitation of the selected algebra and proposes a solution for this.

II. COMPARISON OF POLICY LANGUAGES

Security policy languages have occasionally been compared over time by researchers in order to pave the way for future research or, to help security architects customise their security infrastructures. However, these comparisons have often focused on a small number of policy languages, or are already out-dated due to the fact that, the IT industry is continuously evolving at a rapid pace. From the vast array surveys regarding security policy languages reviewed by this research, we have used the following reports which compared these languages from different perspectives:

A. Scenario-based Comparison
   The ideology of Duma et al report is based on scenarios [4]. To be more precise, in order to compare policy languages the report defines sets of criteria. Following this, for each individual criterion which emerges from real user needs, the report presents a scenario to evaluate the language. If the scenario can be expressed and encoded in a language then the language fulfils the corresponding criterion. This point of view on the comparison of policy languages makes this report unique in its category.

B. Criteria-based Comparison
   De Coi et al. rigorously analysed twelve policy languages over a three years period, with their report being concluded in 2008 [5]. The first criterion which they considered was that the language selected must be popular and widely used. The languages which they reviewed were Cassandra, EPAL, KaoS, PeerTrust, Ponder, Protune, PSPL, Rei, RT, TPL, WSPL and XACML. De Coi et al. evaluated these languages by comparing two sets of criteria namely, core policy properties and contextual properties. Each of these sets were further broken down into sub-categories.

   As noted above, policy languages can be evaluated and classified from different perspectives. However we believe the top level classification of policy languages, as described in [5] and has restated here, fits perfectly in the context of this research.

   Policy languages can be classified into one of three groups namely standard-oriented, research-oriented or “in between” of these two mentioned groups. Standard-oriented
policy languages are well defined and widely shared within the industry. However, they come with a restricted/minimal set of features. Research-oriented policy languages are popular amongst academics. They usually provide advanced features to their users and go beyond the boundaries which have been put in place by standard-oriented policy languages. There is also a third group of policy languages which are neither sufficiently advanced nor fully compatible with standardisation rules to be considered in either of the abovementioned categories. These languages are grouped in the third category referred to as “in between”. We choose XACML from the standard-oriented group, Ponder from “in between” and Protune from the research-oriented group for consideration.

III. ALGEBRA FOR POLICY LANGUAGES

As soon as a combination of heterogeneous security policies became vital, researcher realised that an independent nontrivial combination process, namely an algebra for policy languages, had to be introduced. As a result, a number of policy language algebras have been brought forth. Apart from the fact that using algebra minimises misunderstanding and ambiguity of policies when different parties refer to the same policy [6], algebra can be used for the decentralisation of policy descriptions where complicated and sophisticated policies get broken down into smaller manageable policies [7]. In the context of our research, it was almost obvious from the beginning that using algebra is inevitable as we needed it to describe all policies at an abstract level. As the result we reviewed and critiqued a number of policy language algebras in detail.

We believe the algebra introduced by Rao et al [8] paved the way and could be extended in order to cover all the policy languages we selected in section II. In the following section, we first discuss the policy semantics and then detail the limitation of the algebra before providing an appropriate extension to overcome the restriction.

A. Policy Semantics

Rao et al. presented a simple yet powerful algorithm to describe XACML policy languages in [8]. This was then extended by Zhao et al. in [9]. In their notation, \( A \) is a finite set of \( a \) which characterises an object, subject or an environment. In the same sense, a domain defines a set of possible values and is denoted by \( \text{dom}(a) \).

**Definition 1** Let \( a_1, a_2, ..., a_k \) be attribute names, and let \( v_i \in \text{dom}(a_i) \) \((1 \leq i \leq k)\), \( r \equiv \{(a_1,v_1), (a_2,v_2), \ldots, (a_k, v_k)\} \) is a request.

**Definition 2** A security policy is defined as a request evaluation function \( P : ST \times A \rightarrow D \), where \( ST \) is the set of system states, \( A \) represents a finite set of actions and \( D \) denotes the set of decision tuple for authorisation and obligation associated with: \( P \{Da,Do\} = \{(Y,Y),(Y,NA),(N,NA),(NA,NA)\} \). The function \( P \) takes a system state \( st \in ST \) and an action \( a \in A \) as input, and returns a decision tuple \( (da,do) \) determining whether \( a \) is authorised and obliged to execute in state \( st \). As it is obvious from the decision tuple set, it does not include \( (N,Y),(NA,Y) \) as the obligation state is satisfied with positive authorisation.

**Definition 3** Let \( S \) be the set of subjects, \( T \) be the set of targets, \( E \) be the set of event triggers and \( C \) be the set of conditional constraints. System state is defined as \( ST = E \times C \times S \times T \). This definition allows a system state to be described as \( st = s(e,c,s,t) \) consisting of an event trigger \( e \in E \), the conditional constraint \( c \in C \), subject \( s \in S \) and target \( t \in T \).

B. Policy Constants and operators

In addition the algebra defines its policy constants as follows:

**Permit policy**: \( P_+ : ST \times A \rightarrow \{(Y,Y)\} \). \( P_+ \) authorises all authorisation requests in any state without considering any obligation.

**Deny policy**: \( P_- : ST \times A \rightarrow \{(N,Y)\} \). \( P_- \) denies all authorisation requests in any state without considering any obligation.

The algebra also comes with its operators which are:

**Addition (+)**. Integrated policy \( P_I \) would be union of \( P_1 \) and \( P_2 \).

**Intersection (&)**. Given \( P_1 \) and \( P_2 \), \( P_I \) is defined as the intersection of these two polices if \( P_I \) returns the same decision which is agreed by the two policies.

**Negation (¬)**. \( P_\neg(st,a) \), return \( P_I \) which effectively denies/permits all requests which \( P_I \) permits/denies.

**Subtraction (-)**. \( P_1 \) which denotes the result of \( P_I(st,a) - P_2(st,a) \) is defined as a policy which allows all the requests which are authorised/obliged by \( P_1(st,a) \) and are not applicable by \( P_2(st,a) \).

**Projection (Π)**. Taking into account the fact that the state of an environment is determined by events, constraints, subjects and targets, as described in Definition 3, and assuming \( c \) is a computable subset of \( ST \times A \), the projection operator restricts the policy \( P \) to requests which are satisfied by \( c \).
\begin{equation}
P_I(st,a) = \Pi_{P_{(ST,A)}}^{(da,do)} = \\
\begin{cases} 
(da,do) & \text{if (st,a) } \in c(ST, A) \text{ and } \\
(NA,NA) & \text{otherwise }
\end{cases}
\end{equation}

\textbf{IV. ALGEBRA LIMITATION}

The algebra, as briefly explained above and detailed in [8] and [9], is undoubtedly capable of expressing policy languages which are defined within a domain using a set of ground, i.e. variable free, authorisation and obligation terms. However in this section we would challenge the algebra with more sophisticated policy languages in order to extend the algebra which was presented by Rao et al and introduce a fine grained algebra for policy languages with negotiation capability. As mentioned earlier, extending the algebra presented by Rao can be presented as the contribution of this paper.

\textbf{A. A simple negotiation scenario}

The following example, which is widely shared amongst policy languages with negotiation capabilities such as Protune, goes beyond the concept of unilateral negotiation as we know it in the traditional distributed system. By adopting the approach used by Duma et al in their report [4], we re-evaluate the defined algebra using the following scenario. In the following scenario, Alice, who is a police officer, would like to apply for a free Spanish course online. She does not mind providing information as long as it is not sensitive.

\textbf{Step 1} Alice requests to access eLearn’s free Spanish course.

\textbf{Step 2} eLearn replies by requesting that Alice show a police badge issued by the State Police to prove that she is a police officer, and her driver’s licence to prove that she is living in the same state.

\textbf{Step 3} Alice is willing to disclose her driver’s licence to anyone, so she sends it to eLearn. However, she considers her police badge to contain sensitive information. She tells eLearn that in order to see her police badge, eLearn must prove that they belong to the Better Business Bureau (BBB).

\textbf{Step 4} Fortunately, eLearn has a Better Business Bureau membership card. The card contains no sensitive information, so eLearn discloses it to Alice.

\textbf{Step 5} Alice now believes that she can trust eLearn and discloses her police badge to eLearn.

\textbf{Step 6} After verifying that the badge is valid and that Alice owns it as well as the driver’s licence, eLearn gives Alice the special discount for this transaction [10].

As is apparent from the above scenario, in simple terms negotiations can be divided into a series of steps. A message is usually exchanged in each step whilst the state is partially evaluated. Each evaluation leads the negotiation to the next level. The messages exchanged at each step are of different types, such as “Query message”, for example, Is Alice entitled to a discounted course?, “Policy sets” messages which contain credentials, or simple decision messages which indicate the end of negotiation, possibly with a decision [11].

\textbf{B. Limitation and the solution}

The policy algebra as been described so far is incapable of expressing the above mentioned scenario simply because we have added another dimension to our scenario; negotiation. In the above example, the negotiation process between eLearn and Alice and of course the final decision are unpredictable because Alice is located outside the boundaries of eLearn’s environment and therefore her behaviour is beyond the visibility of eLearn. Hence, the algebra, as defined above, is not capable of formulating the state of environment as stated in Definition 3.

We are now ready to introduce the extended version of the algebra. In order to achieve this goal and formulate negotiation process two steps must be taken.

First, the definition of a security policy i.e. definition 2 needs to be finely modified as follow:

\textbf{Definition 2} A security policy is defined as a request evaluation function \( P : \Sigma \times ST \times A \rightarrow D \), where \( \Sigma \) denotes the finite set of non-ground literal, \( ST \) is the set of system states, \( A \) represents a finite set of actions and \( D \) denotes the set of decision tuple for authorisation and obligation associated with: \( P(Da,Do) = \{(Y,Y),(Y,NA),(N,NA),(NA,NA)\} \). The function \( P \) takes a system state \( st \in ST \), an action \( a \in A \) and non-ground state literals \( \Sigma \) (that intuitively specifies which state literals must be used) as input, and returns a decision tuple \((da,do)\) determining whether a is authorised and obliged to execute in state \( st \). As it is obvious from decision tuple set, it does not include \((N,Y),(NA,Y)\) as obligation state is satisfied with positive authorisation.

Second, we must introduce another operator which utilises both sets of ground and non-ground literal states held at any given time. Assuming that \( ST \) denotes the ground literal states and refers to the set of literals which are held at the current state of environment, and \( \Sigma \) denotes the set of non-
Theorem 1: For all policies $P$

1. In relation to $ST$ and $\Sigma$, policy $P$ has no infinite complete traces.

2. All complete traces of policy $P$ (which are defined as finite sequences of policies with an end policy element of $Pol_i$ with regards to $ST$ and $\Sigma$) have the same final element, that is, policy $P$’s decision.

**Proof:**

1. We must prove A) policy $P$ cannot have a trace with infinite end elements and B) Policy $P$ cannot have infinite traces with finite sets of end elements.

   A) The term **complete trace** used in theorem implies that the trace must come to an end that is, $polcy_n$ (which denotes the final decision of the Policy $P$). In contrast, definition 2 clearly introduces a finite set of decision tuples for any Policy $P$. In other words Policy $P$ cannot have an infinite set of complete traces with an infinite set of decisions/final elements.

   B) Arguably we could have an infinite number of scenarios with a finite number of final elements. However using the term in relation to $ST$ and $\Sigma$ within theorem narrows down the number of scenarios and distinctly specifies which set of ground and non-ground literals is used.

Taking the above facts into account the first part of the theorem proves itself because based on definition 2, policy $P$ cannot have complete traces with infinite end elements $polcy_n$ such that $n\rightarrow\infty$ and at the same time utilising the set of $ST$ and $\Sigma$ narrows down the number of scenarios, hence policy $P$ cannot have infinite complete traces with regards to $ST$ and $\Sigma$.

2. Again using the terms $ST$ and $\Sigma$ implies that the second part of the theorem is referring to a specific scenario. To prove this, we must refer to definition 2 which specifies that the final decision of a policy is determined by three inputs as $\Sigma \times ST \times A \rightarrow D$. In other words, the expression can be read as: As long as combination of $\Sigma$, $ST$ and $A$ satisfies the policy $P$, it will make a decision. The way in which the policy collects this information has no effect on the decision which is made. Considering that trace is a sequence of policies which individually come to a decision and considering the fact that the order of evaluating the sub-policies has no effect on policy $P$’s decision (with regards to $\Sigma \times ST$) proves that different complete traces must have the same final element.

To make this part of the proof more tangible, let us take the abovementioned scenario when Alice asks for a course discount. If we keep the ground and non-ground literals of the environment the same, any alteration to the sequences of the event does not change the policy’s decision. In other words if Alice asks eLearn to disclose their BBB membership number first and then discloses her driving licence and her police badge number, she would still be eligible for a discount.

C. Algebra completeness

Proof of completeness of extended algebra is the next task on our list. We adopt the same approach described in [8] and [9]. In their approach they used a 2-dimensional combination matrix as shown in Figure (2). Bearing in mind the fact that each cell can potentially have four different values namely $(Y,Y),(Y,NA),(N,NA),(NA,NA)$ there will be $4^{16}$ different combination matrices for a 2-dimensional matrix. The authors then proved there is an algebra expression to describe each and every individual policy which is presented by the combination matrix.

<table>
<thead>
<tr>
<th>$P_1$</th>
<th>$P_2$</th>
<th>([N,NA])</th>
<th>([N,NA])</th>
<th>([Y,Y])</th>
<th>([NA,NA])</th>
</tr>
</thead>
<tbody>
<tr>
<td>([Y,Y])</td>
<td>$e_1$</td>
<td>$e_2$</td>
<td>$e_3$</td>
<td>$e_4$</td>
<td></td>
</tr>
<tr>
<td>([Y,NA])</td>
<td>$e_5$</td>
<td>$e_6$</td>
<td>$e_7$</td>
<td>$e_8$</td>
<td></td>
</tr>
<tr>
<td>([N,NA])</td>
<td>$e_9$</td>
<td>$e_{10}$</td>
<td>$e_{11}$</td>
<td>$e_{12}$</td>
<td></td>
</tr>
<tr>
<td>([NA,NA])</td>
<td>$e_{13}$</td>
<td>$e_{14}$</td>
<td>$e_{15}$</td>
<td>$e_{16}$</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2. A 2-dimentional combination matrix**

**Theorem 2:** Let $M_i$ denote combination matrix in sequence $i$ of a complete trace of a given policy $P$ which
requires partial evaluation with regards to $ST$ and $\Sigma$, then an algebraic expression exists which describes policy $P$.

**Proof:** The complete trace of policy $P$ implies that the number of sequences in the trace are finite, ($1 \leq i \leq \infty$). Theorem 3 of [8] also proves that for any combination matrix there exists an algebra expression which contains algebra operators. Hence by adding up all expressions which describe the combination matrices of different individual sequences, we would be able to present policy $P$ using algebra expressions.

What we must consider is the fact that for most cases the minimal set of operators which are needed to describe and formulate policies are $\{P_+, P_-, \cdot, \div, \land, \lor, \Pi\}$. The operator $H$ is needed to describe and divide complex policies such as policies with negotiation capabilities to a finite sequence of sub-policies, in this case the aforementioned operators would be sufficient to describe them in detail.

**D. Algebra expressions**

In practice, expressing a policy using proposed algebra requires multiple operators which must be used simultaneously, and as such we must define the algebra expression. An expression consists of a left associative, an operator and a right associative. Trace has the highest precedence, with negation and projection having the same priority, followed by intersection and addition respectively.

Assuming $P_1 = P_{1}(st,a)$ and $P_2 = P_{2}(st,a)$, the two algebra expressions which extend the expressions already defined in [8] can be presented as:

$$H(P_1 + P_2) = (HP_1) + (HP_2)$$

$$H(P_1 \& P_2) = (HP_1) \& (HP_2)$$

**V. CONCLUSION AND FUTURE WORK**

In this report we have outlined the challenge of translating policies between different policy languages and have illustrated how the management of a distributed network which utilises heterogeneous policy languages is affected by that. Furthermore, we have described an interoperability framework for policy languages to address the issue. It is evident that our proposed framework, as a generic platform, would allow policy languages to be translated to and from each other and studying all available policy languages is unlikely to fit into the context of this research. In light of these facts, we have categorised policy languages into three groups according to their capabilities and have nominated one candidate from each set.

As these policy languages should eventually be mapped onto an abstract layer in order to be translated, we have illustrated how by using algebra at an abstract level we could prove that translation of policy languages to/from each other is possible. Finally, we have also summarised our study on available algebras for policy languages and suggested the one which is most suited for our research. In addition, we have pointed out the limitation of the algebra for policy language and provided our solution for overcoming the restriction.

There are many aspects of future work which have resulted from this paper. We are keen to design and develop a DSL (Domain Specific Language) for policy languages based on the facts we have proved and presented in this paper. The DSL at the core of our framework will facilitate the translation of policy languages to each other. We have already considered appropriate approaches and the development of the DSL is under way.

**REFERENCES**


