Seminar Web Technologies to aid Dominance Detection for Access Control Policies

Jason Barron and Steven Davy
FAME, Telecommunications Software & Systems Group
Waterford Institute of Technology, Ireland
{jbarron, sdavy}@tssg.org

Abstract—We present a dominance detection algorithm as part of a policy authoring process that makes extensive use of semantic models to perform a novel dominance detection of access control policies, where groups of deployed policies are considered in union to discover redundancy. The approach is targeted towards the pre-deployment stage of the policy authoring process and aims to help prevent the introduction of redundant policies into the system. To achieve this, semantic queries are executed over instances of new and deployed policy elements in order to select matching elements for further analysis. The semantic queries may return a large number of deployed policy elements so we present an algorithm that prunes the search space to reduce the problem size. We show that for large sets of deployed policies, we can discover relatively large sets that are considered dominant.

I. INTRODUCTION

Access control policies are used to enforce authorization decisions against requests for access to the resources and services of a domain. Previously deployed access control policies may realise the same or opposing behaviour as a new or candidate access control policy. However, this may not be discovered until after deployment of the candidate policy which has the potential to introduce redundant policies into the system.

Redundant policies have an adverse effect on the performance of analysis and evaluation processes carried out for policies as they needlessly consume system resources and require additional processing time. Additionally, a significant number of redundant policies deployed in a system impacts severely on the processing time for access requests on policy decision points. By analysing policies before refinement, the introduction of potentially redundant policies can be avoided. The dominance detection process outlined in this paper considers if there is a combination of multiple deployed policies that can already realize the behaviour of the new candidate policy. That way, we say that the candidate policy is dominated by deployed policies.

Typical access control policies (i.e. XACML, Ponder, WSPL, etc.) are composed of an arbitrary number of elements (i.e. subject, target, action, conditions, etc). Semantic queries can be executed over instantiations of these elements from a candidate and deployed policies to determine if the policies are specified against the same domain entities. By analysing a policy’s elements, we can ascertain if some form of dominance relationship exists between a candidate and deployed policy over their elements.

We run preliminary semantic queries over the deployed policies to only retrieve relevant policies that mention some terms in the candidate policy. We specified and implemented the policy element match algorithm, that is a modified greedy set cover algorithm, to ascertain if there exists a combination of deployed policies that covers the candidate policy. The domain and policy ontology is an important aspect of our work but is not presented in this paper, as we focus primarily on the policy element match algorithm, its specification and evaluation. The policy element match algorithm is specifically tailored towards matching groups of policy elements from an arbitrary number of policies. Our approach can discover redundancy where an entire set of policies is returned that covers the candidate policy.

The outline of this paper is as follows: §II outlines related work; §III outlines the dominance detection approach. §IV provides some evaluation of the approach. §V summarises the paper and outlines directions for further work.

II. RELATED WORK

Previous approaches to dominance detection [3], [1], [2], are targeted towards detection of inconsistencies over specific low-level policy models (firewall, routing, etc.) that cannot be easily extended to cater for other policy models. Our approach of using ontology models augmented with semantic rules is capable of detecting inconsistencies over various policy models and at different levels of abstraction.

The authors in [8]–[11] all propose methods for detecting redundancy between policies. However, each approach is based on a pair-wise analyses of the policies (or policy sets) which means that these approaches are not capable of detecting redundancy that may occur over groups of policies. However, our approach analyses policies in union in an attempt to detect such occurrences of redundancy.

In [14], the authors propose a conflict free access control model. The model maps every subject to a group and every object to a type. Access requests are based on privileges granted to the group and the requesting subjects role within the group. The authors outline situations in which redundancy can occur and propose to use priorities to resolve redundancy conflicts, but do not provide an implementation of an algorithm to detect such redundancy conflicts.

Most of the work outlined take a pair-wise approach to the analyses of policies to detect dominance between pairs
of policies or policy sets. These approaches could be used to
detect (on a pair-wise basis) the same redundant policies but
would require many iterations (comparisons) to ensure that
the deployed policies cover completely the candidate policy.
This paper builds on our previous work [5], [6] where we
outlined a policy conflict analysis process for the analysis of
newly specified federation policies against previously deployed
(local/federation) policies. This paper extends that work by
newly specified federation policies against previously deployed
policies. This section outlines our policy dominance detection pro-
duced against patterns of policy inconsistencies to reduce the
policy search space and return pertinent deployed policies for
consistency analysis processes required for authoring policies during refinement
of the federation policies.

III. DOMINANCE DETECTION APPROACH

This section outlines our policy dominance detection pro-
duced extensible semantic queries specified against patterns of policy inconsistencies to reduce the
process, depicted in Figure 1 that takes a two phase approach.
In the first phase we utilise extensible semantic queries speci-
ified against patterns of policy inconsistencies to reduce the
the policy search space and return pertinent deployed policies for
analysis. The second phase in the process identifies matches
over an arbitrary numbers of policy elements (both candidate
and deployed policy element sets) which allows us to detect
potential inconsistencies more efficiently than using pair-wise
policy analysis techniques.

![Dominance Detection Process](image)

The policy dominance detection process attempts to dis-
cover if deployed policies either solely, or in combination,
realize the behaviour of the candidate policy. The algorithm
takes as input a set of candidate policy elements (i.e. Subject,
Target, etc.,) and sets of deployed policy elements and returns
a reduced set of covered deployed policy elements that can
then be used as input for further iterations of the algorithm.
The algorithm attempts to reveal cover over one specific policy
element at a time until all the elements of a policy have been
analysed.

A set cover is sought for each element to reduce the number
of deployed policies that can feasibly cover the candidate
policy. If there are no such deployed policies remaining, then
no cover exists and the process exits. Interestingly, along the
way some policies will partially cover the candidate policy.
This information may later be recycled to investigate to what
degree is a candidate policy covered.

A. Policy Dominance Detection

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{C} \sum_{j=1}^{D} c_{ij} x_{ij} \quad (1a) \\
\text{subject to} & \quad \sum_{p \in P} x_{ip} \geq 1, \quad \forall p \in P, \forall i \in C \quad (1b) \\
& \quad \sum_{i=1}^{C} x_{ij} = (0 \text{ or } 1), \quad \forall j \in D \quad (1c) \\
& \quad x_{ij} \in \{0,1\} \quad (1d)
\end{align*}
\]

We specify policy dominance detection as a optimization
problem that aims to discover the minimal combination of
deployed policies that, when considered together, cover the
elements of a given candidate policy. The optimization prob-
lem is described in equation 1 and is similar in form to the
set cover optimization problem. The primary differences are
that there may be multiple element sets related to the candidate
policy, similarly there are multiple element sets related to each
deployed policy.

\( C \) is the number of elements defined for a particular policy
model. \( D \) is the number of deployed policies in a particular
policy based management system. The decision variable \( x_{ij} \)
has an integer value of 0 or 1 and indicates whether a particular
deployed policy element is selected as part of the dominance
detection solution. The objective function aims to minimize the
cost \( c_{ij} \) of including each deployed policy \( x_{ij} \) in the solution
set. The constraints over the decision variable are that for each
element \( (p \in P) \) of the candidate policy, the sum of deployed
policies that include the candidate policy element should equal
to 1 or more. This ensures that each candidate policy element
is covered. Also to ensure that each element of the candidate
policy is covered entirely by the deployed policies, the number
of covers should sum to the number of policy elements if
selected at all, otherwise they should sum to 0.

Solution Space: Calculating the minimum number of de-
ployed policies that overlap to cover a candidate policy is
an \( NP \) complete problem [7]. This is due to the fact that
all combinations of deployed policies need to be considered
together to ensure an optimal solution is found. The solution
space for the problem is therefore \( 2^n \) where \( n \) is the number
of deployed policies. Effectively the solution space doubles
on the addition of each new policy. The approach we take
seeks to discover, to a high degree of accuracy, any possible
combinations of policies that can be considered together
to cover the candidate policy. We term this analysis as a
dominance detection.

B. Policy Element Selection Phase

The policy element selection phase makes use of semantic
queries that are inherently extensible and provide a minimal
form of analysis across all deployed policies in order to reduce
the search space for policy comparison by the policy element
match algorithm thereby increasing the overall performance
of the dominance detection process. Central to the policy
selection process is the use of semantic queries to return a
much reduced set of deployed policies (pertinent to domi-
nance detection) as input to the element match algorithm.
However, other forms of policy inconsistency can easily be
accommodated by the policy selection process as only minimal
modification is required to the semantic queries in order to
return the relevant set of deployed policies for a particular
type of inconsistency check. Our semantic query patterns can
be easily modified and extended to identify various types of
domain independent and application-specific policy inconsis-
tencies (redundancy, conflicts, etc.) defined in the literature.

C. Policy Element Match Phase

The algorithm outlined in Algorithm 1 is used to create
a list of related policies. The algorithm first attempts to find
an identical match between the policy element identifiers from
each set (both candidate and deployed) and if a match is found
between the policy elements they are added to the relation-
ship list. Once all identical matching policy elements have
been identified, the algorithm attempts to union deployed policy
elements to discover if the union of partially matched policy
elements matches the candidate policy element. If a match
is discovered between the union of deployed policy elements
and candiate policy element, the deployed policy elements
are added to the relationship list and associated together. The
reason for associating the union of deployed policy elements
is that any future analysis on these policy elements would
have to consider these policy elements together. The algorithm
continues to union partially matched deployed policy elements
until no more policy element matches can be identified.
The final step of the process, is to intersect the policy element
set identifiers and if a deployed policy has a policy element
in each set then this deployed policy (or union of deployed
policies) matches the candidate policy. The policy author is
notified regarding the list of matched deployed policies and
can make a decision regarding the deployment of the candidate
policy.

The list \( d_{list} \) contains identified matched deployed policies
and is initially set to zero. The set \( U_d \) contains the set of
remaining unmatched deployed policy elements. The set \( U_c \)
contains the set of candidate policy elements for the algorithm
to match against. The set \( d_c \) contains at each step the identi-
ﬁed matched policy elements and may hold partially identi-
ciﬁed matches. When the inner loop is entered the maximum subset
S is chosen from the set \( U_d \). This maximum matched subset
S is then removed from set \( U_c \) and placed in set \( d_c \). If the
subset S only partially matches the identiﬁers of the set \( U_c \),
that partially matching identiﬁer is removed from the set \( U_c \)
while the algorithm attempts to discover if other subsets of S
can match the remaining identiﬁers in the set \( U_c \).

If there are deployed policy elements remaining in the set
\( U_d \), the algorithm attempts to union the remaining policy

<table>
<thead>
<tr>
<th>Algorithm 1</th>
<th>Element Match Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>element-Match:(CandSet, DepSet) \to DepSet</td>
<td></td>
</tr>
<tr>
<td>element-Match(c, d) \equiv</td>
<td></td>
</tr>
<tr>
<td>( d_{list} \equiv 0 )</td>
<td></td>
</tr>
<tr>
<td>( U_d \equiv d )</td>
<td></td>
</tr>
<tr>
<td>( d_p \equiv 0 )</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>( U_c \equiv c )</td>
<td></td>
</tr>
<tr>
<td>( d_c \equiv 0 )</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>select an ( S \in U_d ) max</td>
<td>( S \cap U_c )</td>
</tr>
<tr>
<td>( U_c \equiv U_c \cdot S )</td>
<td></td>
</tr>
<tr>
<td>( d_c \equiv d_c \cup {{S}} )</td>
<td></td>
</tr>
<tr>
<td>if ( ( S \equiv 0 \text{ and } U_c \neq 0 ) )</td>
<td></td>
</tr>
<tr>
<td>( d_p \equiv d_t )</td>
<td></td>
</tr>
<tr>
<td>( d_c \equiv 0 )</td>
<td></td>
</tr>
<tr>
<td>while ( ( S \neq 0 \text{ or } U_c \neq 0 ) )</td>
<td></td>
</tr>
<tr>
<td>( U_d \equiv U_d \cdot d_c )</td>
<td></td>
</tr>
<tr>
<td>( d_{list} \equiv d_{list} \cup {d_c} )</td>
<td></td>
</tr>
<tr>
<td>while ( ( d_c \neq 0 ) )</td>
<td></td>
</tr>
<tr>
<td>return ( d_{list} )</td>
<td></td>
</tr>
</tbody>
</table>

IV. EVALUATION AND ANALYSIS

Our prototype implementation includes the creation of on-
tology models that are used to represent both the structure
and behaviour of a domain at multiple levels (one for each level
in the organization) and is based on a modified version of the
algorithm used by Barrett et al. [4].

SPARQL [13] a semantic query language was used to query
the policy knowledge base and return the policy element IDs
along with the policy IDs. Jena [12] was used to load the
required domain and policy ontologies, issue semantic queries
over the loaded ontological knowledge bases and store the
results in a data structure. The policy element match algorithm
that was implemented in the Java programming language.

We conducted a number of experiments to determine what
impact an increase in the number of matched union policy
elements has on the performance of the policy element match
algorithm. In our scenario, this relates to pre-existing policies
deployed for a large organization where individuals already have access control policies defined for them, and a new group policy is being deployed. The results will show how the number of individual workers can have an impact on the algorithm. For one such experiment, a single candidate policy element set was input into the algorithm. The number of matched deployed policy element sets would need to be considered in union to cover the candidate policy element set for this experiment.

The number of union policies was initialised at two and increased to a maximum of 100 deployed policies. The results are depicted in Figure 2 and indicate that the time required to detect dominance increases marginally as the number of union deployed policy element sets increases. This is due to the complexity of maintaining the identified matches between the policy element sets from multiple distinct deployed policy element sets. In real terms, the results show that when a new candidate policy is being deployed to cover a large set of individuals with pre-existing policies, it takes longer to ascertain a cover, or dominance detection which can be expected.

Note, that the greedy algorithm is known to find a solution that is $0.58 + \ln(i)$ times the optimal solution, where $i$ is the size of the largest set [7]. Therefore, the optimal solution is not guaranteed to be found. This impacts on our solution in that there may be sets of deployed policies that better cover the candidate policy; however, we do not care about the best cover, only that there exists a deployed set of policies that together have the same behaviour as the candidate policy.

V. CONCLUSIONS

Policy dominance detection is a novel approach to reduce the occurrence of redundancy that harms policy performance, due to complex interrelated policies. The policy element match algorithm outlined in this paper can be used to provide such policy dominance detection. The overall effect will be to notify a policy author who can then take a decision to proceed with the policy deployment or not. It is clear from this work that the cover between policy elements (where policy elements need to be considered in union to realise the behaviour of a candidate policy) has a marginal impact on the performance of the policy element match algorithm future work will investigate methods for improving the processing time of the algorithm by possibly caching previously detected cover between policies.

ACKNOWLEDGEMENT

This research activity is co-funded by Science Foundation Ireland via the FAME strategic research cluster (http://www.fame.ie), grant no. 08/SRC/I1403, the Irish Research Council and by Marie Curie Actions under FP7.

REFERENCES