Strategies for Reducing Risks of Inconsistencies in Access Control Policies

Bernard Stepien[1,2], Stan Matwin[1,2,3], Amy Felty[1,2]
[1]School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada
[2]Devera Logic Inc., Ottawa, Canada
[3]Institute for Computer Science, Polish Academy of Sciences, Warsaw, Poland
(bernard | stan | afelty)@site.uottawa.ca

Abstract—**Managing access control policies is a complex task. We argue that much of the complexity is unnecessary and mostly due to historical reasons. There are number of legacy policy specification languages that all have limitations of some kind. These limitations have forced policy implementers to use certain styles of writing policies, often resulting in inconsistencies. The detection and resolution of these inconsistencies has been widely researched and many solutions have been found. This paper highlights new possibilities for avoiding inconsistencies, drawing on the expressive power allowed in the condition field of rules in modern languages such as XACML. In particular, we show that making use of this expressive power has many advantages—it allows organizations to considerably reduce the number of policies and rules required to protect company assets; it provides improved views and summaries of related policies; and it allows increased scalability of analysis tools, such as tools that detect inconsistencies and tools that perform audits to verify compliance to regulations. Such tools are increasingly important in the current environment where the number of regulations governing company security continues to grow. In addition, we show how our user-friendly representation for the XACML language facilitates the use of complex conditions by increasing their readability. This increased readability has the additional benefit of allowing non-technical users to better understand the implementation of their policies. These factors all contribute to a lower risk of inconsistencies in policies.**

Keywords-access control management; policy conflict detection; policy management workbench; XACML

I. MOTIVATION

Access control (AC) policies are specified using a variety of languages that have been created either by vendors for specific applications or by standardization bodies in order to alleviate some of the limitations of vendor languages. Some of the languages are proprietary, and some of these eventually became de-facto standards, while others have been subject to intensive standardization efforts. The history of this evolution can be found in [1]. The main characteristics of these languages can be summarized as follows:

- Number and organization of variables used to specify various criteria
- Available data types for variables
- Mechanisms for specifying values
- Degree of complexity of conditions

All of these characteristics contribute to a language's ability to specify coarse-grained vs. fine-grained rules. The requirements of security regulations make fine-grained policies more attractive, but because most legacy systems use coarse-grained policies, a number of awkward techniques have been developed in order to achieve fine-grained policies using legacy languages that support mainly coarse-grained policy writing. These techniques result in programming styles for rule conditions that have been widely documented as leading to increasing the risk of introducing conflicts. A number of algorithms for resolving policy conflicts have been proposed for various AC languages. They differ in the programming techniques or formalisms used. For instance, [2] uses an algorithm that can be implemented in any general purpose language, [3] uses Alloy, [4] uses a CLP Prolog based expert system, [5] uses an algorithm formally verified by a theorem prover, and [6, 7] use XML tools. They are mostly language specific and so far no attempt has been made to find solutions across languages.

While AC literature often focuses on firewalls, we have worked extensively with other types of applications, e.g. controlled credit cards, controlled access to parts of documents, controlled cell phones, etc. In all of these applications, the introduction of complex conditions using large sets of variables has led to increased security.

This paper has the following contributions: firstly, we review the current methods for conflict detection in rule-based policies, especially in the context of XACML. Secondly, we argue for the need for a user-friendly non-technical notation (and interface) to define and verify policies. Such a notation makes it possible to easily use complex expressions in the condition part of the rules– without such complex conditions the equivalent "simple" rule sets get large and difficult to build and explain. Thirdly, we show how such complex conditions in XACML lead to more compact rule sets, which can be built and understood by policymakers themselves, without relying on specialized IT personnel. Fourthly, we demonstrate how the use of complex conditions leads to a very efficient implementation: the encoding of the rules in Prolog, combined with Prolog's advanced backtracking mechanisms, results in a very efficient method of checking rule sets for inconsistencies.

In *Proceedings of the Fifth International Conference on Availability, Reliability, and Security (ARES)*, February, 2010.

II. ACCESS CONTROL POLICY SPECIFICATION LANGUAGES

Access control specification languages can be classified into two broad categories:

- Application specific languages
- Generic languages

A. *Application specific languages*

Application specific languages are usually based on the following basic principles:

- A fixed number of variables are pre-defined for the application.
- It is possible to specify one or multiple values for a given variable. This includes ranges of values or, less commonly, sets of values. We call such a specification of value or values for a particular variable an *atomic expression*.
- Conditions are always an implicit conjunction of atomic expressions.

This class of languages includes various proprietary access control languages such as Cisco IOS for firewalls [8] and representations for Windows firewalls [9].

B. *Generic languages*

Generic languages are by definition not tied to any specific application. The basic principles of generic languages are:

- Unlimited user-specified variables
- Complex conditions using full and unlimited complexity of logical expressions

Complex conditions imply the capability to use both conjunctions and disjunctions on variables. The XACML language [10] falls into this category.

C. *Language editing capabilities*

Access control languages have evolved over the years. Early languages featured low-level instructions that were hard to read and were entered in command line mode or plain text editors. Such editors had neither syntactic nor semantic checking. Modern languages often come with full graphical user interfaces (GUIs) where each variable is clearly separated from one another and a number of value selection widgets are available. This kind of feature reduces the risk of errors, especially on restricted domains. Such GUIs, by definition avoid the need for syntactic checking but semantic checking is still required.

D. *Expressiveness of languages*

A policy or rule is usually addressed to a specific *target*, and a target is itself composed of a subject, a resource, an action and an obligation. However, especially in role-based policy languages, a subject may play different roles and ambiguities can arise that may result in conflicts. For example a manager may be authorized to perform an action while an employee may not. Since a manager is also an employee, this leads to a natural ambiguity. Some of this ambiguity can be resolved by adding more attributes on which a finer set of criteria can be applied.

Languages based exclusively on conjunctions of atomic expressions need to specify separate rules for what would naturally be a disjunction. As a matter of fact a policy or rule set is a natural disjunction. Each rule is an alternative to other rules. Thus, languages based exclusively on conjunctions and single or limited combinations of values for a variable require using special strategies to reduce the number of policies or rules to describe a configuration. The most common strategy consists of using a combination of two kinds of rules, broader rules and exception rules, with opposite effects. An *effect* is either "deny" or "permit".

For example, the Cisco IOS firewall language with a fixed number and order of variables enables the implementer to specify either exactly one protocol at a time, or use the value *ip* that encompasses all protocols. Thus, the implementer may define a rule to deny access to any *ip* protocol, and then define another exception rule for a single protocol and port number that permits access for this selected protocol and port only. Since Cisco firewalls search the rules in order and use the first applicable one, the exception rule must be placed before the broader rule.

```
acl 101 permit tcp any host 11.22.33.44 9000
acl 101 deny   ip  any host 11.22.33.44 9000
```

Also, the capabilities of AC languages for specifying multiple values most often allow the specification of only a single range. Thus the specification of a rule that encompasses several ranges must either be broken down into several rules each with a single sub-range, or must use a broad rule with one or many exceptions. The same problem arises with combinations of variable values. For example, the specification of an AC rule for two different protocols and two different destination ports would require the following four rules:

```
acl 101 permit tcp any host 11.22.33.44 9000
acl 101 permit tcp any host 11.22.33.44 8000
acl 101 permit udp any host 11.22.33.44 9000
acl 101 permit udp any host 11.22.33.44 8000
```

These kinds of problems can be solved with languages like XACML that allow the expression of complex conditions. In this case, disjunctions involving a single variable could be used in order to specify that the rule is effective for different values for that variable.

Another distorting factor, and consequently a frequent source of errors, is the use of default rules. Some languages like Cisco IOS default to a deny effect. Others such as XACML do not provide defaults, but instead allow a default policy to be specified by the implementer using a rule without a target or a condition. It is to be noted that when such default rules are specified, all other active rules should have the opposite effect; any active rule that has the same effect as the default is by definition redundant. However, such redundant rules are necessary when specifying exceptions. The exception rule has to be specified before the broader rule in a context where the first applicable rule is always applied.

In some languages, the concept of exception rule is central. For example, MS-Windows firewalls exclusively specify exceptions, since the basic default effect is deny. Optimizing the specification of exceptions has been addressed in [11]. However, in a language with the capability to express complex conditions with multiple ranges in a single rule, the full extensions to language expressiveness that they propose would not be necessary. In XACML, one can easily specify exceptions by using only two rules, one for the main specification and the other for the exception. In contrast legacy languages need one rule per range and per class of effect (permit and deny).

More problems arise when default rules specify different effects for different policies such as could happen in XACML. Also, the absence of default rules can result in no rule being applicable. Since a decision by PDPs (Policy Decision Points) will be to deny the access; this absence is a kind of implicit default rule. This however has the advantage of avoiding explicit conflicts between an exception rule and the default rule, thus avoiding false positives.

For both historical reasons and language expressiveness reasons, the migration to more expressive languages is becoming more frequent as shown for example in [12]. However, a direct translation from one language to another is only the first step. Ideally, the features of the new language should make it possible to reduce considerably the number of policies and thus make them more manageable, as discussed in Section IV.

III. CHARACTERISTICS OF ACCESS CONTROL RULE CONFLICTS

The risk of policy conflicts has been widely recognized. The difficulty of avoiding conflicts among policies or rules within policies can be best illustrated by the recommendation to use single-effect policies found in the Fedora user manual [13]. Even more interesting is the substance of the recommendation. They mention that this strategy will result in writing more individual policies, but that this is preferable from a policy management point of view. Although, this recommendation highlights an attempt at defining strategies to minimize the risk of conflict by using the features of an AC language, in this case we actually observe a form of regression to practices that were common in older languages. This is mainly due to an increased number of rules—which is precisely what modern languages are supposed to avoid. Another example illustrating awareness of the conflict problem is the fact that modern languages such as XACML have attempted to incorporate a mechanism to resolve conflicts, showing that the designers consider such conflicts to be inevitable, in this case due to the distributed nature of policies. However, as we discuss below, this feature does not always lead to a satisfactory solution and may in fact cause more problems.

Inadequacy of conflict resolution algorithms

XACML allows the resolution of conflicts at run time based on an administrator-defined rule-combining algorithm. When a set of rules matches a given access request and there is a conflict between a pair of rules in the returned set, the rule combining algorithm indicates which rule effect should prevail. The risk of error is by definition 50% and can produce false positives or false negatives. This kind of solution to the conflicting rules problem, however, is necessary in the context where policy sets may be distributed and thus could be under different authorities that may not coordinate their policy decisions. This inherent risk emphasizes the need for analysis that detects conflicts, reports them and eventually fixes them, either on a periodic or on-going basis. This class of problems has been explored in [14].

While modal conflicts (values that satisfy two rules with opposite effects) are relatively straightforward to detect, they are considerably more difficult to detect in role-based systems because such systems rely on the fact that a user can have different roles and thus can have different permissions with potential opposite effects depending on the role they take [15]. For the role-based policy cases, conflict resolution is more important than modal conflict detection and results in models that include priority variables as in [16] and PDPs that contain elaborate conflict resolution mechanisms. However, in [16], they report that even role-based systems that use priorities to resolve conflicts are prone to errors and require conflict detection mechanisms.

IV. STRATEGIES TO REDUCE THE RISK OF RULE CONFLICTS

While policy conflicts are inevitable, their frequency can be influenced using various strategies that include either specification language features or analysis approaches.

A. Using non-technical notations and related tools

Access control notations have been historically based on two variants:

- Notations where variable names are absent and are resolved using a fixed order for variables
- Notations that use explicit variable names

The first category of languages requires that the implementer have technical knowledge of which field corresponds to which variable. This can sometimes be obvious due to the presence of specific values, but there are times when ambiguities cannot be avoided. For example, Cisco IOS distinguishes between specification of a single host and specification of multiple hosts using bit masks. The keyword *host* is used to resolve the syntactic ambiguity. However, technical knowledge is required to specify variable values.

Complex technical notations such as XACML use explicit variables names but are difficult to use by non-technical users because of the lengthy XML tags and complex constructs described by the corresponding highly technical XML schema. While, there is no doubt about the advantages of XACML, it is important to alleviate this problem in one of two ways:

- The use of GUIs
- The use of user-friendly non-technical interface notations in combination with GUIs

The use of GUIs

The prime purpose of GUIs is to avoid requiring the implementer to have extensive technical knowledge. There are basically two categories of GUIs:

- Primitive value selection GUIs
- Language based GUIs

GUIs with primitive value selection capabilities allow a user to select values for predefined variables and by definition do not allow room for creative and complex rules. They are available for proprietary AC languages such as Cisco IOS. They are also very often available for proprietary solutions within a company, and thus not available to a general audience.

Language-based GUIs allow a user to build a specification expressed as a domain-specific language. They help the user by both suggesting the syntax of the language and selecting values for variables. One of the benefits of such languages is to reduce the requirement for the user to have extensive domain knowledge as much as possible by providing both lists of allowed values and lists of operators for constructing more ad hoc complex conditions.

However, in most cases, this class of GUIs still requires knowledge of language features, knowledge of domains of operators, etc. The most advanced language-based GUI implementation is XACML studio [17]. The technical knowledge requirement problem still arises when attempting to construct a complex condition. In this case, XACML studio uses the XML hierarchical model and is not very helpful in providing information about what to use at a given level of hierarchy in the tree. In addition, the condition that is constructed is displayed in full XACML syntax with all its verbosity.

The use of non-technical notations

The results of a survey [18] point to the critical gap that exists between non-technical policy makers and highly technical policy implementers. The main concern expressed in this survey is the fact that policy makers cannot understand the technical implementation of their policies. This problem is further aggravated by the techniques implementers have to use to alleviate the limitations of a given language. These techniques result in non-intuitive constructs for the policy maker. Policy makers are not the only ones facing this problem. There are new access control applications and approaches—for example in ubiquitous systems—that allow end-users who are not policy makers to define their own policies [19]. Their policies will sometimes intentionally conflict with policies set up by higher authorities. The reason for this is that they take into account ad hoc context that would make the higher authorities' policies a security risk. In such applications the distinction between policy maker and policy user is rapidly dissolving. Also, it is commonly expected that policy users are equally as non-technical as policy makers.

In [20] we have attempted to create a presentation notation that eliminates the need for knowledge of most technical aspects of a language, like syntax or even semantics. This presentation notation also provides various levels of overview where several variables and their values within a rule or several rules can be viewed at the same time. This overview feature is a determinant factor in avoiding conflicts. This is because conflicts are often introduced when conflicting values of another rule are not visible at the time of coding a new rule. This occurs, for example, when there are lengthy domain tags in the way or when rules are scattered in a rule base. The use of a notation that is close to the implementation language has another advantage—it is self-documenting. Both the policy maker and the policy implementer see the same text and thus do not need to translate their representation back and forth to each other's language. This is in contrast to other approaches such as [21] that merely translate natural language into XACML. Our presentation notation does not fully eliminate the need for technical knowledge but it separates the concerns of a technical administrator that sets up the configuration of an application and the non-technical user (policy maker or end user) that writes the policies. The latter has no need to know the configuration details. The examples shown in this paper use our notation. Also, this non-technical notation does not eliminate the need for a traditional value selection GUI. The selection of variables and values is still handled as in a traditional GUI, but the display of a complex condition and its manipulation is achieved through use of our non-technical notation.

Our proposed notation is only a display notation. It is neither a new language nor a replacement for XACML. This notation is based on the following basic principles:

- Stay as close as possible to the user's natural language by avoiding any technical terminology for operators and maintaining the overall structure of a natural language.
- Offer an implicit structuring by organizing the natural language into a tree so as to avoid ambiguities about scope of operators. Indentation defines the scope.
- Organize the tree so as to make it consistent with the natural language statement of the condition by using an infix representation for conjunction and disjunction operators.
- Maintain XACML's natural non-binary nature of conjunction and disjunction operators but eliminate its original list representation.
- Use a different, yet still casual terminology for conjunction and disjunction operators depending on their position in the tree hierarchy.
- Ensure a full graphical overview of the expression being built at all times regardless of its complexity. This implies preventing collapse of portions of the tree.

B. The use of complex conditions

The prime advantage of using complex conditions is the capability to combine several rules into one single rule. This immediately reduces the number of rules required to describe a policy. There are several related advantages of this principle:

- Avoid the scattering of related rules.
- Provide a natural overview on conditions and exceptions.

For example, the Cisco IOS example shown previously could be represented with a single rule using XACML's ability to express complex conditions. It would be represented in our non-technical XACML notation as follows:

```
Effect: permit
Condition:
          protocol is one of tcp, udp
       and
          destIp is 11.22.33.44
       and
          destPort is one of 8000, 9000
```

The above example uses disjunction on values for a single variable that is really represented internally as a disjunction on equalities for a single variable.

Another important fact is that while most legacy languages use top-level conjunction between variables, modern languages allow the specification of top-level disjunctions between different variables as well as values. This introduces new capabilities of factoring based on disjunction. For example, in a controlled credit card application, the specification of the condition that food can be purchased only on Mondays or Wednesdays while travel can be purchased only on week-ends can be specified in a single rule instead of two rules using a complex condition with a high-level disjunction. In the following example the emphasis is put on the kind of merchandise purchased. Thus, complex conditions enable the emphasis to be put on different topics and thus be closer to a policy maker's preoccupations.

```
   Merchandise is food
      provided that DayOfTheWeek is one of
                                 Monday, Wednesday,
or
   Merchandise is travel
      provided that DayOfTheWeek is one of
                                   Saturday, Sunday
```

It is interesting to note that the above high-level disjunction could be rewritten in a different way. This other form provides a rule grouping where the day of the week is the central structure, as opposed to the above rule where the merchandise to be purchased is the central structure.

```
   DayOfTheWeek is one of Monday, Wednesday,
      provided that Merchandise is food
or
   DayOfTheWeek is one of Saturday, Sunday
       provided that Merchandise is travel
```

Disjunctions enable the user to put the emphasis on different variables. The groupings using disjunctions better express a thematic structuring of policies. In our example we emphasize correctly that the purchase of different goods is mostly governed by the day of the week. This is different from the usual declarative approach of traditional access control languages. Even though this form does not result in a more compact representation, we believe it leads to a more comprehensible one, by grouping together policies that belong together semantically.

A condition is naturally hierarchical. Sub-constraints are stepwise refinements of the parent constraint. This is the essence of fine-grained policies.

The use of such non-technical notation in combination with the use of disjunction opens a different style of policy composition. Instead of adding new atomic rules preferably at the end of a policy, the administrator or user could perform this addition in two steps:

- Query the policy base for rules that contain values for a set of variables appearing in the rule to be added.
- After viewing the results of the query, choose the appropriate existing rule to modify, taking into account the requirements of the new rule.

The above described procedure naturally increases the chances of preventing conflict. In particular, since the user is forced to consider the existing related rules, this makes him aware of the potential for conflicts.

We leave as an exercise to the reader the computation of how many combinations of legacy style rules the following complex example would require using our non-technical notation:

```
   Merchandise is clothing
      provided that DayOfTheWeek is one of
        Friday
          provided that TimeOfTheDay before 19:00,
        Saturday
          provided that TimeOfTheDay between
                                      12:00 and 18:00
or
   Merchandise is food
      provided that DayOfTheWeek is one of
        Wednesday
          provided that TimeOfTheDay between
                                    16:00 and 18:00,
        Thursday
          provided that TimeOfTheDay between
                                      12:00 and 13:45
or
   Merchandise is travel
      provided that DayOfTheWeek is Sunday
      provided that BalanceOfAccount over 1000.00
```

The use of complex conditions as we are proposing is in some way a departure from the XACML model. There is no doubt that XACML can be used to describe complex conditions, but initially XACML did not intend conditions to be used in that way. In XACML, the basic use of target descriptions was supposed to be sufficient. Conditions were invented only to handle time or date condition requirements. Targets are used in XACML mostly to ease storage and retrieval of policies into databases. In fact, the XACML standard is somewhat confusing since the XACML schema allows the same kind of expressions in both targets and conditions. Consequently, this could mean that complex expressions could be introduced in targets. However, this practice has never been implemented.

However, there are characteristics of the XACML target that are a potential source of confusion. Subjects, resources and actions have two different levels of specification. For example individual subjects in a subjects list are considered as a disjunction while the SubjectMatches inside a given subject are joined by an implicit conjunction. It is difficult even for a technical person to have an overview of the logical meaning being expressed in a XACML target. After all, conjunction and disjunction operators are a central concept of natural languages.

The concept of target (used even in legacy languages because it is so natural) is somewhat restrictive when considering the use of disjunctions. Disjunctions can be used at the target level, but this implies that the condition that follows applies equally to all elements of the disjunction, while one would like to specify different conditions for each element of the disjunction. For example the specification of different conditions for different resources can be achieved only by using separate rules when specification is done using targets as follows:

```
Rule #1 target resource is Projector
Condition:
time between 09:00:00 and 17:30:00

Rule #2 target resource is Computer
Condition:
   time between 00:00:00 and 23:59:59
```

These rules can be expressed as a single rule when moving the resource from the target to the condition:

```
Single rule #1 no target
Condition:
Resource is one of
   Projector
      provided that time between 09:00:00 and
                                          17:30:00,
   Computer
      provided that time between 00:00:00 and
                                           23:59:59
```

Thus, we propose an approach that migrates some of the target content to the expression of complex conditions in order to resolve the inadequacies resulting from the use of XACML targets and thus allow more flexibility in the construction of complex conditions.

C. *Using static modal conflict detection strategies*

While so far we have proposed manual solutions to reduce the occurrence of policy conflicts, this does not eliminate the need for automated conflict detection methods. Actually, while conflicts can be reduced by reducing the number of rules and avoiding scattering of conditions, more complex rules can be more difficult to compare to each other and thus bring their own contribution to conflicts. Modal conflict detection is nowadays considered as straightforward. Consequently, we do not focus on yet another conflict detection algorithm but instead on how to make best use of automated detection of policy inconsistencies.

Inconsistencies can be detected at two different stages:

- At compile time by comparing the values of criteria specified explicitly.
- At run time by comparing the values that originate in the environment and that can not be known at compile time.

There are two ways to detect static or compile time conflicts:

- Global conflict detection where a rule base is entirely searched by comparing all pairs of rules for every policy.
- Local conflict detection where a new rule being entered is compared immediately to the already existing rules.

The first strategy is unavoidable when importing a legacy rule base, while the second strategy is more appropriate for entering new rules. Combinations of both strategies can be used. Global conflict detection becomes a one time event when importing an existing rule base, while local conflict detection is performed only when adding a new rule or modifying an existing rule.

D. *Using modal conflict detection techniques for auditing*

Performing modal conflict detection may reveal inconsistencies in policies but does not guarantee that the policies reflect the intention of policy makers. However, we have found that the principles of modal conflict detection can be used for two purposes other than conflict detection itself:

- Querying policies for audit purposes
- Checking conformance to higher-level policies

Querying policies for audit purposes

An AC query can be viewed as a high-level rule. An audit query is usually targeted to a subset of variables and conditions. For example, the question of determining under which conditions someone has access to a specific resource can be expressed as a single rule. This rule can be used by a conflict detection mechanism that will compare it to all rules in the rule base. The simple rule would encode the information needed to identify the resource either by using a single value or by using a more complex expression that would express a range of values for one or a small number of variables. Also, the effect can be set according to the purpose of the query. For example, a query about what policies grant access to a building during night operation would consist of performing a redundancy check (looking for the policies that permit such an action) between the following simple rule and all the policies in the policy set:

Query condition:

```
   Building is "building_1"
and
   TimeOfDay between "20:00:00" and "06:00:00"
```

The results would include all the rules that specify more fine-grained conditions and have values for variables that intersect with this general rule. For example, the above rule

has values intersecting with the following more complex condition of another rule.

Query result:

```
   Building is "building_1"
and
   Role is one of scientist, maintenance
```

The above rule illustrates the kind of results one can expect from an audit. The detected rule does not specify any time constraints while the audit rule does not specify anything about the role. Thus, the auditor can evaluate the relevance of the rules that would be returned by such a query.

Also, by merely changing the effect of the query, we can obtain the list of rules that prevent access to the resource. Thus the principle of modal conflict detection is a natural auditing technique.

A visual inspection by the access control policy implementer or policy maker could spot rules with more complex conditions that were initially overlooked simply because of the physical scattering of the rules in the policy base.

Using conflict detection algorithms for auditing is more efficient than data base searches because data base searches can not consider complex expressions. They are usually limited to exact matches on targets rather than evaluation of conditions.

We are using an expert system written in Constraint Logic Programming (CLP) Prolog for our modal conflict detection tool using techniques that extend the ideas in [4]. Although a query-based system suggests an expert system approach, modal conflict detection can be implemented in any general purpose programming language with various degrees of effort depending on the features available in the language.

Checking conformance to higher-level policies

Higher level company policy is often expressed as a set of "golden rules." Conformance to this policy can be performed by encoding the golden rules and performing modal conflict detection between the golden rule policy set and the targeted policy set. In a sense, this is similar to the above audit techniques except that here the conformance checking is performed using a set of rules instead of a single rule and using only opposite effects.

E. Scalability and performance issues

For very large policy bases, scalability and performance may be an issue when the number of policies becomes large. Both strategies described above require having all policies in memory. This is a very common way to handle the problem. This requirement actually comes from the traditional operation of PDPs that load a policy set at initialization time. This is not always the case, however, since an increasing number of PDPs use data bases to extract only policies that match a given target. Another attempt to alleviate scalability problems is by using distributed PDPs as shown in [22]. While most of the time, reading all rules into memory at once does not present scalability issues, it is the comparison of several policy sets that may encounter such problems. However, a divide and conquer approach can be used especially when rules are stored in data bases where their retrieval can be optimized. For example, retrieval can be optimized in XACML by using database queries on target elements (subject, resource and actions). Policies and their rules can be clustered into related targets prior to executing the conflict detection on complex conditions, which are difficult to use as clustering criteria.

However we determined that the best way to avoid scalability problems is by using complex conditions. This is primarily because as we have shown above, this reduces the number of rules. The impact of this is that it directly reduces the quadratic number of comparisons between rules. In addition, an interesting consequence of using an expert system written in Prolog is its use of backtracking, which avoids exhaustive pair-wise comparisons of atomic rules.

We discovered this solution to scalability while experimenting with generators that produce large numbers of policies. This generator generated rules as conjunctions of variables where each variable contains an equality on only one value (no ranges) at a time. Rules for all combinations of values for all the variables were generated. Thus, if there are n variables $v_1,\ldots,v_n$ that each have a number of values $n_1, \ldots, n_m$, the resulting number of combinations are the product of their numbers of values:

Number of combinations = $n_1 \times n_2 \times \ldots \times n_m$.

These conditions have the following structure:

```
   Var1 is v11           Var1 is v1nm
and                   and
   …            …        …
and                   and
   Varn is vn1           Varn is vnnm
```

We have determined that in this particular case all of the rules resulting from these combinations can be reduced to only one single rule by using disjunctions on values of each variable such as:

```
   Var1 is one of  v11, …, v1n1
and
   …
and
   Varn is one of  vn1, …, vnnm
```

The most surprising result was the performance of the modal conflict detection on this modified form. For example for 4116 rules, the modal conflict detection execution time decreased from 183 seconds for the individual combinations rules to only 2 seconds for the single rule, detecting the same 2058 conflicts in both cases. This is due to the fact that Prolog's backtracking avoids comparing the n-j variables' values repeatedly as is done when performing pair-wise comparison of atomic rules even when using Prolog. The

new approach also uses Prolog's optimization of value searches through hash tables and other indexing methods. A similar method was implemented in [2] where a rule base was organized as a tree-based filtering representation. Our single rule with disjunction of values in the condition is precisely a tree. The difference is that their method is used only at conflict detection time because Cisco firewall rules do not allow the use of complex conditions, while our method is used to permanently restructure the rule base itself even before the conflict detection procedure is applied to it.

Thus, the strategy to use more complex conditions has positive effects both at the human level, where more complex conditions can prevent errors due to scattering of information, and at the machine analysis level, where fewer complex rules reduces the scalability problem that arises from pair-wise comparisons.

F. Our policy management workbench

Our policy management workbench has five main features:

- A PAP that uses our non-technical notation
- An internal representation of policies based on XACML
- A translator to and from selected languages, including XACML to our internal representation. Policies are still stored externally in XACML format
- A modal policy conflict detection checker based on Prolog-CLP expert system principles
- A policy audit feature based on the modal conflict detection feature

Using the XACML structure as a common internal representation is a solution with minimal risk. Since XACML is a generic language that allows an unlimited number of variables to be considered, there always exists a method for mapping a given language to XACML. This is illustrated by the modeling of attribute release policies (ARP) and the somewhat related attribute acceptance policies (AAP) to XACML found in [23].

V. CONCLUSION

Our research has shown that currently, access control management is in a state where many languages and techniques are in use concurrently, and that these techniques are not always efficient and often lead to inconsistencies. We have shown that there are critical advantages to using more complex conditions in order to better structure policy bases as well as to help both policy makers and implementers compose such policies. Future work will include studying how these new principles can be applied to role-based and prioritized rules. Another interesting area of future work is the design of algorithms to automatically convert a rule base with atomic conditions and numerous rules to one with fewer rules containing complex conditions.

REFERENCES

[1] R.Batouba and I. Aib, "Policy-Based management: A Historical Perspective," IEEE Transactions on Network and Service Management, Vol. 4, 2007.

[2] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. "Conflict classification and analysis of distributed firewall policies," IEEE Journal on Selected Areas in Communications, 23(10):2069-2084, Oct. 2005.

[3] M. Mankai and L. Logrippo, "Access control policies: Modeling and validation," in K. Adi, D. Amyot, and L. Logrippo, Eds., 5th NOTERE Conference (Nouvelles Technologies de la Repartition), pp. 85-91, Gatineau, Canada, 2005.

[4] P. Eronen and J. Zitting, "An expert system for analyzing firewall rules," 6th Nordic Workshop on Secure IT Systems, pp. 100-107, 2001.

[5] V. Capretta, B. Stepien, A. Felty, and S. Matwin, Formal correctness of conflict detection for firewalls," Proc. of the 2007 ACM Workshop on Formal Methods in Security Engineering, Nov. 2007.

[6] K.Fisler, S.Krishnamurthi, L.Meyerovich, and M.C.Tschantz, "Verification and change-impact analysis of access-control policies," Proc. of the 27th International Conference on Software Engineering, 2005.

[7] V. C. Hu1, E. Martin, J. Hwang, and T Xie, "Conformance checking of access control policies specified in XACML," Proc. of the 31st Annual International Computer Software and Applications Conference, vol. 2, 2007.

[8] J. Sedayao, Cisco IOS Access Lists, O'Reilly, 2001.

[9] Microsoft firewall configuration GUI, http://www.microsoft.com/-windowsxp/using/networking/security/winfirewall.mspx

[10] XACML, OASIS standard, http://www.oasis-open.org/committes/tc_home.php?

[11] J.G. Alfaro, F.Cuppens, and N.Cuppens-Boulahia, "Management of exceptions on access control policies," IFIP International Federation for Information Processing, Springer, Boston, 2007.

[12] G. Karjoth, A. Schade, and E. Van Herreweghe, "Implementing ACL-based policies in XACML," Annual Computer Security Applications Conference, 2008.

[13] D. Davis, Fedora commons, Fedora XACML policy writing guide, http://www.fedora- commons.org/confluence/display/FCR30/-Fedora+XACML+Policy+Writing+Guide

[14] P.Mazzoleni, B.Crispo, S. Sivasubramanian, and E. Bertino, "XACML policy integration algorithms," Proc. of the Eleventh ACM Symposium on Access Control Models and Technologies, 2006.

[15] B.J. Garback and A.C. Weaver, "XACML for RBAC and CaDABRA: constrained delegation and attribute-based role assignment, unpublished, University of Virginia, 2005.

[16] F.Cuppens, N. Cuppens-Boulahia, and M. Ben Ghorbel, "High-Level conflict management strategies in advanced access control models," Proc. of the First Workshop in Information and Computer Security, Electronic Notes in Theoretical Computer Science, 186:3-26, Jul. 2007.

[17] XACML studio, http://xacml-studio.sourceforge.net, 2005.

[18] L. Bauer, L. Faith Cranor, R.W. Reeder, M. K. Reiter, and K. Vanica, "Real life challenges in access-control management," 27th CHI Conference, 2009.

[19] M.White, B.Jennings, and S. van der Meer, "User-centric adaptative access control and resource configuration for ubiquitous computing environments," 7th International Conference on Enterprise Informaition Systems, 2005.

[20] B.Stepien, A.Felty, and S.Matwin, "A non-technical user-oriented display notation for XACML conditions," E-Technologies: Innovation in an Open World, Proc. of the 4th International MCeTech Conference, Springer, 2009.

[21] C.A. Brodie, C.-M. Karat and J. Karat, "An empirical study of natural language parsing of privacy policy rules using the Sparcle policy workbench," Proc. of the 2nd Symposium on Usable Privacy and Security, 2006.

[22] V.Dhankar, S. Kaushik, D.Wijesekera, and A.Nerode, "Evaluating distributed XACML policies," Proc. of the 2007 ACM Workshop on Secure Web Services, Nov. 2007.

[23] W.Hommel, "Policy-based integration of user and provider-sided identity management," International Conference on Emerging Trends in Information and Communications Security, 2006.