DYNAMIC ADAPTATION OF ACCESS CONTROL POLICIES

Vijay Bharadwaj and John Baras
Institute for Systems Research
University of Maryland
College Park MD 20742

ABSTRACT

We describe an architecture and algorithms for deriving an access control policy by composing access control requirements specified at multiple levels in a command hierarchy. Our method can detect conflicts in requirements, and find a policy that maximally satisfies the requirements, by satisfying higher priority requirements at the expense of lower priority ones. It also allows for easy verification of the final policy by an administrator. The architecture allows quick adaptation of policies to changing situations, by providing for delegation of authority while ensuring that high priority requirements will always be satisfied.

INTRODUCTION

Modern military operations increasingly depend on the rapid deployment and secure operation of exceedingly complex computer systems and networks, involving tens of thousands of users and an even greater number of data objects and applications. Such systems need clearly defined access control policies which define the privileges of each user with regard to each data or program object, in accordance with overall doctrine and mission objectives. However, due to the large number of variables and interdependencies involved, it is neither efficient nor practical to specify such detailed policies manually. To make matters worse, a single mission may require the cooperation of multiple units or policy domains, each with their own systems and policies. Further, military policies are generally specified through a hierarchical chain of command; commanders at the top of the hierarchy specify doctrines and missions in general terms, and commanders at lower levels refine this into a specific policy in accordance with their specific needs and capabilities.

Current methods and systems for access control often do not meet these needs. Access control policies must be specified manually at a very detailed level, often in a language that is not easily accessible to the average user. Few automated tools exist for composing or refining policies, or for checking that a given access control state accomplishes the objective it is supposed to. It is hard to adapt policies to changing mission needs; the entire policy may have to be changed, and this must be done through the same manual process used for specifying the original policy. These problems are exacerbated when coalition operations are involved. In that case, each coalition member may not even be willing to completely divulge their internal access control policies, or the capabilities and limitations of their access control systems.

In this paper we describe an architecture and algorithms that allow an access control state to be automatically computed by synthesizing component policies specified at different levels of detail. This work arose from our previous research on dynamic coalitions [1, 2]. In our framework, mission and operational requirements are treated as constraints on the final access control state. These constraints are specified at varying levels of abstraction by human administrators. Software agents then combine them with the existing access control policies and hierarchies to derive the access control state that is maximally consistent with all the constraints. This is done through the use of soft constraints, which mark some constraints as more important than others and express hints to the system regarding which states are more desirable than others.

In the following sections, we summarize some prior work on access control and constraint programming. We then describe our proposed architecture, and how such a system can adapt to changing requirements. Finally, we present the algorithms for deriving access control policies in such an architecture, along with some illustrative examples.

0-7803-8140-0/03$17.00 (C) 2003 IEEE

RELATED WORK

An access control policy consists of all the rules that a system uses to control access to its resources. It includes a definition of the objects to which access is controlled, the mechanism for controlling access, and the rules for determining whether or not to grant an access request. Thus, in large systems, access control policies can be very complicated.

An access control model is an abstract mathematical representation of an access control system. Military systems commonly use a multilevel Mandatory Access Control (MAC, [3]) model. In such a model, each object and each subject is assigned an access class consisting of a security level and a set of categories. The security level is drawn from an ordered set, such as $< Top\_Secret, Secret, Confidential, Unclassified >$. The set of categories is a subset of an unordered set and is assigned on a need-to-know principle. An access class A is said to dominate another class B (denoted $A \geq B$) if A has a security level at least as great as B and A's categories include those of B. Thus the access classes form a lattice under the dominance relation. These classifications are used to restrict the access privileges of subjects to objects - the exact semantics depends on whether the system is intended to protect secrecy or integrity.

Moffett and Sloman [4] were among the first to discuss the problem of automated management of large distributed systems. They define policy hierarchies, where the abstraction levels of policies increase as we go up the hierarchy. They explore the refinement of policies in such a system, where high level policy is set by managers, and this policy is refined and implemented at lower levels by human or automated agents. The authors also discuss the use of Prolog to determine whether a set of lower level policies completely satisfy a higher level policy without any conflicts between subjects. However, they did not discuss the application of their work to a practical system.

Constraint solving has been an active area of research in Artificial Intelligence. A Constraint Satisfaction Problem (CSP, [5]) consists of a set of problem variables, a domain of possible values for each variable, and a set of constraints, each of which specifies an acceptable combination of values for one or more of the problem variables. Therefore in a CSP, each constraint is simply a set of tuples over some subset of the problem variables. A solution for a CSP is an assignment of values to the variables that satisfies all the constraints of the problem.

Semiring-based CSPs (SCSPs, [6, 7]) are an extension of CSPs wherein the constraints are not Boolean but defined over an appropriate semiring. In this way SCSPs are able to model soft constraints (i.e. preferences), partial knowledge and prioritized constraints.

A semiring is a tuple $< A, +, \times, \mathbf{0}, \mathbf{1} >$ where

- A is a set with $\mathbf{0}, \mathbf{1} \in A$;
- $+$, the additive operation, is closed, commutative and associative over A with $\mathbf{0}$ as its identity element;
- $\times$, the multiplicative operation, is closed and associative over A with $\mathbf{1}$ as its identity element and $\mathbf{0}$ as its absorbing element;
- $\times$ distributes over $+$.

An lc-semiring [6] is a special kind of semiring which can represent a complete distributive lattice; the $+$ and $\times$ operations of the semiring correspond to the *lub* and *glb* operations of the lattice. Thus the $+$ operation defines a partial order $\leq_S$ over the set A; we say $a \leq_S b$ if $a + b = b$.

A semiring-based constraint system is a tuple $< S, D, V >$ where S is a semiring, D is a finite set and V is an ordered set of variables. A constraint over such a system is a tuple $< def, con >$ where $con \subseteq V$ is known as the type of the constraint, and $def : D^k \rightarrow A$ (where k is the cardinality of V) is the value of the constraint. Thus def assigns a value from the semiring to each combination of values of the variables in con. This value can be interpreted as a strength of preference, a probability, a cost, or something else depending on the problem. An SCSP is then a tuple $< C, v >$ where $v \subseteq V$ and C is a set of constraints.

The solution of an SCSP is the constraint obtained by combining all the constraints in the SCSP and projecting it over the set v of variables of interest. The best level of consistency (blevel) of the SCSP is the projection of the solution over the empty set. Thus the blevel represents the highest valuation that can be attained by a tuple under the constraints. In other words, the blevel gives the maximum extent to which a given set of constraints can be satisfied.

SCSPs have also been used in a variety of applications. For instance, Bella and Bistarelli [8] used them to model the Needham-Schroeder protocol and showed that the model can be used to "discover" a well-known attack on this protocol.

Constraint Logic Programming (CLP, [9]) incorporates the notion of constraints into Logic Programming, by replacing term equalities with constraints, and unification with constraint solving. This allows much more concise representation of problems; it also allows for more efficient implementations of constraint solvers, as it provides additional information that helps guide the search for a solution.

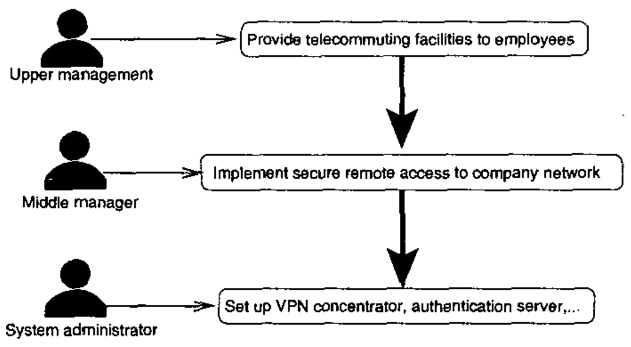
Upper management
Provide telecommuting facilities to employees
Middle manager
Implement secure remote access to company network
System administrator
Set up VPN concentrator, authentication server,...

Figure 1: Policy refinement in organizations

Semiring-based CLP (SCLP) generalizes CLP to soft constraints. The syntax and semantics of SCLP programs are described in [10]. Briefly, an SCLP program consists of a set of clauses of the form $H : -B$. We say this clause holds in an interpretation I iff for any ground instantiation of H, say $H\theta$, we have $I(H\theta) \geq_S I(\exists B\theta)$.

PROPOSED ARCHITECTURE

In large organizations, various requirements, specified at various levels in the command hierarchy, impose constraints on the access control state. Usually, more general constraints are imposed by higher levels in the hierarchy whereas details are specified at lower levels. For instance, in corporations, policies such as "no employee may perform both ordering and procurement functions" are generally specified by upper management whereas details such as "file orders.xls is related to purchasing" are filled in by lower level employees. This process is illustrated in Figure 1.

In our system, access control policy is developed through a multi-step process of composition as shown in Figure 2. Each level in the command hierarchy has an administrator. At each step, this administrator specifies a set of constraints arising from requirements at that level and a set of tests on the final access control state. These constraints and tests are translated into SCLP clauses and goals respectively, and communicated to a central server. The server combines the constraints into a complete SCLP program, which is then solved to obtain an access control state. The different tests are then carried out on the state, and results returned to the administrators, who may then specify more constraints and tests. In later stages, administrators can also retract previously specified constraints. The process concludes when all administrators send a null message to the server, signifying that they have no more tests or constraints to specify. The server then communicates the access control state to the entities responsible for access control in the domain.

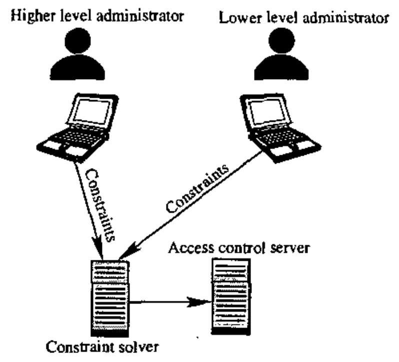

Figure 2: Policy composition architecture

This architecture allows for rapid reconfiguration of the access control state when requirements (and hence constraints) change. When this happens, the administrators engage in a process similar to that described above, except that constraints are added to or retracted from the previously existing SCSP. Such changes may require revocation of privileges already held by users; in some systems, such as those using certificates for access control, this can be a significant problem. However, this is beyond the scope of this paper. Some issues relating to certificate revocation are discussed in [11].

CONSTRUCTING THE ACCESS STATE

The problem of constructing an access state can be formulated as follows. Given a set of access classes A, a set of objects O and a c-semiring $\mathcal{V} = < V, +, \times, 0, 1 >$, we want to construct a scheme to assign access classes to objects. The scheme must allow multiple administrators to specify constraints over this semiring, and then compute the optimal access state satisfying these constraints. We assume that access classes are assigned to users and subjects by some other mechanism.

We observe that when a set of constraints is constructed from different sources as above, there may not exist a unique access control state satisfying all the constraints. In some cases, there may be many such states; in this case the system must pick the best one. Other times, constraints may conflict with each other, such that no access control state satisfies all of them.

We overcome these problems by attaching a priority (taken from $\mathcal{V}$) to each constraint. The system tries to satisfy high priority constraints first. During the multi-step process described above, administrators can add or retract constraints to guide the server to a better state.

Given A, the set of access levels, and O, the set of objects to classify, we express the problem as an SCSP over the constraint system $< \mathcal{V}, A, \mathcal{O} >$, where $\mathcal{O}$ is a set of variables containing one variable for every object in O. We proceed as follows.

- Choose a semiring $\mathcal{V}$ for prioritizing constraints. The semiring is chosen so that the minimal element of $\mathcal{V}$ represents the highest constraint priority.
- Let the highest level administrator specify a set of constraints on the final access state. Each constraint associates the value 1 (the maximal value of the semiring) with tuples of variable values that are acceptable and its priority $v \in V$ to the rest.
- Proceed to the next highest administrator, and so on.
- Solve the resulting SCSP. The blevel gives the priority of the highest priority unsatisfied constraint - a blevel of 1 means all constraints were satisfied. The maximal solution of the SCSP gives the required classification.

The choice of $\mathcal{V}$ will depend on the application. For example, a reasonable choice might be to use the access class of the administrator specifying the constraint as its priority. However, the rest of the algorithm is not altered by the choice of $\mathcal{V}$.

In practice, the SCSP and its constraints are specified as predicates in an SCLP language. In our initial implementation of this concept, we are using the $clp(FD,S)$ [12] system, which provides support for user-defined semirings. So far, we have implemented some simple scenarios similar to those described below.

Note that by using SCLP, we also obtain the capability to verify the algorithm's output. By formulating appropriate goal clauses for the constraint solver, an administrator can run sanity checks to see if the final state satisfies some desired property. Also, we can allow for hierarchical policy refinement.

More formally, we assume that the domain and the objects in it are divided hierarchically into parts, with a corresponding hierarchy of administrators, as in Figure 3. The highest level (Level 1) administrator sets the high-level policy, much like in Moffett and Sloman [4], by defining a few high-level predicates, and leaves some of the predicates undefined. The level 2 administrators then define these predicates and state some high-level objectives for the level 3 administrators, and so on. At the lowest level, constraints are defined on the variables in $\mathcal{O}$.

We use a logical language with the following useful features added to $clp(FD,S)$:

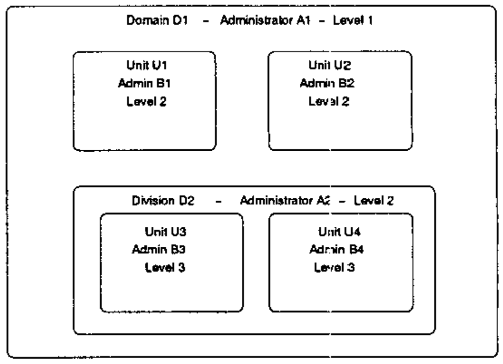

Figure 3: Hierarchical administration

1. **Constant Symbols:** Every member of A, the set of access classes.
2. **Variable Symbols:** We have finite domain variables ranging over A. In particular, there is one such symbol per object to be classified.
3. **Predicate Symbols:** The predicate `dominates/2` expresses the dominance relation on A. The predicate `classify_up/3`, which takes an object (*Obj*), an access class (*Class*) and a semiring value (*Val*) as arguments, is defined to yield a constraint which assigns semiring value 1 to all assignments which classify *Obj* in a class that dominates *Class*, and *Val* to all other assignments. The predicate `classify_down/3` is similar, except it assigns 1 to assignments that classify *Obj* lower than *Class*. In addition to these, a number of application-specific predicates are also used.

Since our language does not contain any function symbols, we have from [10] that once the program is fixed, the value of any goal is computable in time that is finite and bounded by a constant. Thus our language is efficient.

EXAMPLE: CLASSIFYING OBJECTS IN A MULTILEVEL MAC SYSTEM

To illustrate our techniques, we now show how to solve a classification problem given in [13]. As mentioned in the previous section, we could use the access level of the administrator who specifies a constraint as the priority of that constraint. However, in order to achieve a minimal classification, we also need to prioritize states that assign lower classification levels to objects while satisfying the constraints.

Consider the lc-semiring $\mathcal{V} = < V, +, \times, 0, 1 >$ where A represents the set of all possible access classes, and 0 and 1 denote the highest and lowest access classes respectively. Now define the $+$ operation as follows: for two access classes a and b having classification levels l_a and l_b and access groups S_a and S_b, $c = a + b$ is the access level with classification level $min(l_a, l_b)$ and access groups $S_a \cap S_b$. Similarly, $d = a \times b$ is the access level with classification level $max(l_a, l_b)$ and access groups $S_a \cup S_b$. Thus c is the highest access class dominated by both a and b, whereas d is the lowest access class that dominates both a and b.

Now consider the semiring $\mathscr{W} = < V^2, +_2, \times_2, 0_2, 1_2 >$, where V^2 is the Cartesian product of V with itself, while $+_2$, $\times_2$, 0_2 and 1_2 are simply componentwise extensions of the corresponding elements of $\mathscr{V}$. Then $\mathscr{W}$ is also an lc-semiring [6]. Now we associate each assignment of access classes to objects with an ordered pair $< v_1, v_2 >$ where v_1 is the access class of the highest level administrator whose constraint it violates, while v_2 is the highest access class it assigns to an object. Due to the inverted nature of the semiring we have chosen, this means that a maximal solution will be one which tries harder to satisfy higher level administrators, while also trying to avoid overclassifying data.

It is easy to see that basic constraints (lower bound and upper bound constraints) are easily expressed using `classify_up` and `classify_down` respectively. Expressing more complicated constraints from [13], such as inference and association constraints, is also fairly straightforward, if a bit messy to include here.

It is worth mentioning that once the constraints from our example are translated into an SCLP, the semiring can be changed completely without altering the program. So if we wanted to find the maximally closed solution that satisfied a set of constraints, all we would need to do is to exchange the action of the $+$ and $\times$ operators on the second component of $\mathscr{W}$. In this configuration, the constraint solver would try and make sure that no object was underclassified. This ability to switch from a "trusting" to a "paranoid" mode of operation without the need for rewriting the policy may in itself be a useful feature in some situations.

CONCLUSION

In this paper we described an architecture and mathematical framework for composing access control policies and to allow for quick adaptation of policies to changing requirements, as well as verification of the resulting policy. The framework is very flexible; we can accommodate both MAC and RBAC systems by simply configuring some parameters. We investigated the automated negotiation of access state in RBAC systems in a separate paper [14].

Many questions remain open. This paper did not address in detail the computational issues related to the constraint solver and with regard to incremental solvers. We also did not address user interface issues in this paper. Finally, we are still working on extending the framework to incorporate more general operators for composing policies, such as those defined in [15].

ACKNOWLEDGMENT

This research effort was sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-2-0510. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

REFERENCES

[1] V. D. Gligor, H. Khurana, R. K. Koleva, V. G. Bharadwaj, and J. S. Baras. On the negotiation of access control policies. In B. Christianson et al., editors, *Proceedings of the 9th International Security Protocols Workshop, Cambridge, U.K., April 2001*, volume 2467 of *Lecture Notes in Computer Science*, pages 188–201. Springer Verlag, 2002. Also see transcript of discussion, pp. 202-212.

[2] V.G. Bharadwaj and J.S. Baras. A framework for automated negotiation of access control policies. In *Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, 2003.

[3] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report ESD-TR-278, vol. 1, The MITRE Corporation, Bedford, MA, 1973.

[4] J. D. Moffett and M. S. Sloman. Policy hierarchies for distributed system management. *IEEE JSAC Special Issue on Network Management*, 11(9), 1993.

[5] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, NJ, 1995.

[6] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of the ACM*, 44(2):201–236, March 1997.

[7] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. *Constraints*, 4(3), 1999.

[8] G. Bella and S. Bistarelli. SCSPs for modelling attacks to security protocols. In *Principle and Practice of Constraint Programming - CP2000 Workshop on Soft Constraints, Singapore*, 2000.

[9] J. Jaffar and M.J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[10] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint logic programming: Syntax and semantics. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(1):1–29, 2001.

[11] H. Khurana. *Negotiation and Management of Coalition Resources*. PhD thesis, University of Maryland, 2002.

[12] Y. Georget and P. Codognet. Compiling semiring-based constraints with *clp(FD,S)*. In M. Maher and J-F. Puget, editors, *Proceedings of the 4th International Conference on the Principles and Practice of Constraint Programming (CP98), Pisa, Italy, October 1998*, volume 1520 of *Lecture Notes in Computer Science*, pages 205–219. Springer Verlag, 1998.

[13] S. Dawson, S. de Capitani di Vimercati, P. Lincoln, and P. Samarati. Maximizing sharing of protected information. *Journal of Computer and System Sciences*, 64(3):496–541, May 2002.

[14] V.G. Bharadwaj and J.S. Baras. Towards automated negotiation of access control policies. In *Proceedings of the Fourth International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, Lake Como, Italy, June 2003.

[15] P. Bonatti, S. de Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 5(1):1–35, 2002.