

Towards A Mechanism for Discretionary Overriding of Access Control

Position Paper

Erik Rissanen, Babak Sadighi Firozabadi, Marek Sergot

March 24, 2004

1 Introduction

Traditional access control models either permit access or deny it completely. There is an implicit assumption that all access needs are known in advance and that the conditions of those needs can be expressed in machine readable form. However, reality is dynamic and unanticipated situations occur all the time. Most manual administrative procedures are flexible enough to allow people to stretch the rules. In contrast, most access control models for electronic systems do not allow for similar flexibility. A pedantic access control system can cause frustration, or worse, can prevent people from doing their job. In [1], for instance, it is suggested that if a secretary cannot forge the signature of his/her boss in an electronic system, the electronic system is not going to work. An alternative would be to delegate the required power to the secretary explicitly, but this is just another form of planning in advance. Recent works in distributed access control models [2, 3, 4] suggests mechanisms to allow delegation and grant of authorisations when they are needed. We might also imagine a solution in the form of a system for requesting access rights as needed.

However, all these approaches require that some authorisations are created in advance. Either the explicit permission or a delegation right that allows the permission to be created when it is needed. None of these approaches address the issue that it is not always possible to plan ahead, and that the authority empowered to create an authorisation may not be available at the time access is needed and authorisation for it is missing.

What we propose instead is a system that would routinely allow access, under specified constraints, even in the case when it is not explicitly permitted. Instead the system depends on audit and sanctioning for enforcement. This mirrors manual organisations. Established organisation theory recognizes such dependency on rule bending, [5].

Of course such procedures are not appropriate in all situations. For instance, some information has to be secret and trusted only to a select

group. This is also the case in manual organisations: some rooms are locked and require special key cards to allow people in. Our proposed model makes the same distinctions.

We propose to explicitly, in the security policy, distinguish between what a principal *can* do, what it is *permitted* to do, and what it is *forbidden* to do. The importance of these distinctions in computer security is argued in [6]. The intersection of *can* and *not permitted* is what we refer to as *possibility-with-override* or (sometimes) *ability to override*.

We also propose the concept of *authority resolution* which is a process which automatically finds an access control policy authority who can approve a specific override.

We believe that by including override and authority resolution in the access control policy directly, management of overrides can be improved.

1.1 Related Work

The idea of being able to override denied access is not new. Here we mention just some of the recent work.

Dean Povey, [7, 8], has done work that is similar to ours . Povey recognizes the problem with legitimate demand for access in unanticipated situations. His main focus is on guaranteeing system integrity by means of transactions that can be rolled back. We believe that automatic recovery in general is not possible and in many cases costly to implement, so we put our emphasis on audit and manual recovery. However, the issues are orthogonal: automatic recovery, as Povey demonstrates, may be possible in some cases, and here it can be used in conjunction with our methods.

Gunnar Stevens and Volker Wolf [9] have performed a case study at a steel mill. They differentiate between *ex-ante*, *uno-tempore* and *ex-post* access control, depending on whether the permission is granted before, during or after access. Stevens and Wolf noticed that the manual processes used by the steel mill was a mixture of all three types of access control. This compelled them to design a computerized access control system that would allow such a mixture. In relation to their work, this paper deals with *ex-post* access control.

Jaeger et. al. [10] introduce a concept called *access control spaces*. This concept is used primarily for analyzing conflicts in access control policy or to analyze whether a set of assigned permissions and constraints on possible assignments completely cover all possible assignments. The relation to our work is that access control spaces, which present a partition of permissions similar to which we present, can be used to eliminate any ‘forgotten’ access possibilities. However, there is nothing access control spaces can do for those cases where the desired policy cannot be expressed in the given policy language. Jaeger et. al. in fact suggest the use of access override and audit in some cases.

XACML [11] includes the concept of obligation which can be used for instance for specifying different access levels and that an access should be logged.

We have not been able to find any previous work on automatic authority resolution.

2 Motivation

Existing access control systems implicitly assume that all knowledge about the access needs of users are known, and defining an access control policy is ‘simply’ a matter of specifying that knowledge in a suitable formal language. To leave that assumption, we need to think about what kind of knowledge we have and what we cannot have.

Assume that we are to draft an access control policy for an organisation. In order to do so, we can study how the organisation is supposed to function, and define the conditions and subjects of access to specified objects. From this we can gather a set of access rules, which we can encode in the machine readable security policy.

However this list of access needs is going to be incomplete. There are two sources of incompleteness. The first is that not all situations for an access permission will be expressible in the machine readable language in use; the second is that there are many needs we simply cannot predict.

This incompleteness in the possibility of drafting a security policy will present itself as a conflict between confidentiality/integrity on the one hand and availability on the other. For instance, we want to protect the confidentiality of medical data, but we are also afraid of emergencies where unanticipated access might be needed.

To solve this conflict, we want to extend our access control model such that we can leave the decision to access at the discretion of the user. We let the user have the possibility to override a denial of access, within specific bounds. The user is better capable of judging the situation in an unanticipated event than the access control system is. We do not place the whole burden of enforcing the security policy on the access control mechanism alone. Instead, to combat abuse, we log all uses of such access overrides in a special way. Note that this is not the same as logging any access for a post review, as it is usually done.

We note that, while designing an access control policy, we will likely find situations where we can say with certainty that access should not be permitted. The access control model should support this so that access override would not always be possible.

3 Overrides

We can divide all possible future access scenarios into the following categories:

1. *Anticipated, allowed and machine encodable*: Situations for which we can say ahead of time that access should be allowed and for which we can express the conditions in machine readable form. Ex. “All employees can read the company newsletter.”
2. *Anticipated, denied and machine encodable*: Situations for which we can say ahead of time that access should be denied and for which we can express the conditions in machine readable form. Ex. “Non-medical personnel may not read patient records.”
3. *Anticipated, allowed and not machine encodable*: Situations for which we can say ahead of time that access should be permitted but we cannot express the conditions in machine readable form. Ex. “In case of an emergency, any doctor may read the patient’s records.” (We cannot always formally define “an emergency”.)
4. *Anticipated, denied and not machine encodable*: Situations for which we can say ahead of time that access should be denied but we cannot express the conditions in machine readable form.
5. *Unanticipated*: Situations that we have forgotten to consider or cannot predict.

When defining the security policy, we need to partition the “situation space” into these categories, and then map the categories into the security policy. To support the ambiguous cases, we allow three possible outcomes to an access request: *permitted* access, *denied* access and access *possible with override*.

Those situations that we can describe will be easy to include in the *permitted* and *denied* access categories. The situations where we have only a partial description of the situation are placed in the *possible* category. How the forgotten and unanticipated situations are classified depends on how we describe the security policy. There are three alternatives:

1. Define the *permitted* accesses and the *possible* accesses. By default everything else is *denied*.
2. Define the *permitted* accesses and the *denied* accesses. By default everything else is *possible*.
3. Define the *denied* and *possible*. By default everything else is *permitted*.

For instance, in the first case we could say that a doctor is *permitted* to read only the records of his own patients, so we include a permission for that only. However, we can imagine that in case of an emergency a doctor may need to access the records of other patients as well, but there is no way we can define all emergencies formally and in advance, so we include a possibility-with-override for doctors to read any patient record. By default anyone else is denied access and cannot override.

There is a difference between the cases. We cannot describe the scenarios that we do not know in advance, so they will fall in the default category. The most sensible default depends on the organisation. In some cases it may be sensible to protect confidentiality by making denied access the default. In other cases availability may be more important, so possible-with-override is the better default. The third case, where everything is permitted by default, may seem counterintuitive, but might have value in cases where availability is critical.

Jaeger et. al. [10] suggest a categorization similar to the second type, where they allow an user to override some denied accesses.

3.1 Access Control Policies

In this paper, we will focus on the first case listed above, that is when the access control policy defines permitted and possible accesses.

To include override in an access control model we introduce the *possibility-with-override* concept, which is analogous to a permission with two differences. The first difference is that using a possibility-with-override to gain access should be different for the user than using a regular permission. Some kind of warning message should be presented. The other difference is that uses of possibilities should be logged and thoroughly audited.

Most access control policy languages can probably be easily extended to support both and languages already support more than a yes or no answer to an request. For instance, in XACML [11] an obligation could be used to implement both the warning and the logging.

However, in order to assist the audit, we would also like each override to be automatically reported to an authority of that particular permission. This leads to *authority resolution*.

4 Authority Resolution

We call the mechanism by which we from an access control policy can find a set of authorities for a given permission *authority resolution*. The permission does not necessarily have to exist in the policy, in which case we want to find all principals who would be authorised to create the permission. Note that we do not assume that all authorisations are created by a single administrator, but instead we imagine that administration of authorisations

is decentralised, with different administrators/managers controlling different parts of the policy.

The authority resolution mechanism can be used in conjunction with possibilities to assist in the audit of overrides. We (informally) suggest the following approach:

1. The user tries to access an object.
2. The application uses an access control decision function to check whether access should be permitted.
3. The access control decision function responds that the user is not permitted to access, but can override that to gain access.
4. The application asks for a confirmation from the user and the user confirms.
5. The application performs the access.
6. The application uses the authority resolution function to get a set of authorities whom to report the override to.
7. One or more of the authorities audit the override and do any sanctioning ex-post.

To answer the question of who is a *legitimate authority* we note that an approval of an override is in effect a retroactive issuing of a permission that would have permitted the access. Thus *the authorities who should be able to approve an override are precisely those who can create a permission for the access that was overridden*. How to calculate who they are depends on the access control policy language. We will need a language that contains meta level authorisations, that is permissions for changing the access control policy itself. Some examples of such languages are

In some access control policies it is possible that there are multiple legitimate authorities for approving a given override. This leads us to ask whom among these we should notify and in case we notify multiple authorities, in what order and what happens in case some of them approve and some of them disapprove.

From an organisational point of view we can notice that most organisations are more or less decentralised in that there are vertical and horizontal separations of authority. By horizontal separation we mean that different managers have different areas of authority. For instance the manager of the sales department has different authorisations than the manager of the engineering department. By vertical separation we mean that there typically are different levels of management and higher level management commonly delegate authority to lower levels of management. Even though high level

managers in principle have power over low level managers, they may not be familiar with the details of day to day operations. (On a side note, a superior person in an organisational hierarchy does not necessarily have all the authorizations of her subordinates, which somewhat confuses the “vertical” and “horizontal” analogies.) We therefore find two desirable properties for the authority resolution algorithm.

4.1 Desirable Properties

We desire the authority resolution and override approval mechanisms to have

- *Safeness*: Only legitimate authorities should be included.
- *Unobtrusiveness*: Among the legitimate authorities, we should start the audit by notifying those who are most likely to understand the override and least likely to be bothered unnecessarily.

The first property should be easily achieved given an access control model.

For the second property there are more considerations. Mainly we need to find some order in which the authorities should be notified to minimize hassle for high level managers. But we also need to consider what we should do in case several authorities have differences in opinion. Let us look at the order of notification first.

4.2 Order of notification

First we note that the person who created the possibility-with-override that made an override possible is a prime candidate to be notified first, as long as he is a legitimate authority. The rationale is that whoever made the override possible is best placed to judge whether to approve the override.

Now, depending on the particular access control framework, there may be additional information available. If the access control framework has some kind of hierarchy of authorisations, the hierarchy can be used to order the authorities. One example of such a framework is the calculus of privileges, [4, 12], in which all authorisations are issued in the form of delegations, and the delegations form a hierarchy with higher level managers at the top. Such high level authorities should be notified after lower level authorities. Another example would be a directory tree in which authorisations can be inherited along the nodes. Someone authorised for a more specific subtree should be notified before others, since we as a heuristic may assume that he is more familiar with the details of that particular subtree.

It should be noted that depending on the access control framework the ordering of the authorities may not be complete.

4.3 Conflict resolution

Now, assuming we have some (perhaps incomplete) ordering of the authorities, what would we do in case authorities we notify do not take action or differ in their opinions?

Again, this depends on the particular access control framework. We would like to mimic the semantics of the particular underlying framework in use as much as possible. So since we can view the approval as a retroactive creation of a permission, we have to ask the question, what would happen if the authorities would have created different permissions for an access corresponding to the override? This will depend on the conflict resolution policy of the framework.

Consider the case in which the access control framework does not have negative permissions. In case some authorities approve and some disapprove, the approvals should have precedence, since there would be no way to create a negative permission in the first place. If all of them disapprove (or do not care), we view the override as disapproved. This means that the order in which we notify the authorities is not critical as long as we notify all of them until someone approves.

In case there are negative permissions in the access control framework, depending on the conflict resolution algorithm, an authority who also could issue a negative permission blocking the overridden access, would be able to disapprove an override and stop it from being sent to higher level authorities.

Thus, how many of all the authorities should be notified depends on the semantics of the particular access control framework in use.

5 Future work

We have started work on incorporating these ideas in the “calculus of privileges” access control framework, which supports the management structure of authorisations in the form of a hierarchy of delegations. We will present an algorithm for authority resolution based on that framework and prove some properties of that algorithm.

References

- [1] A. M. Odlyzko, “Economics, psychology, and sociology of security,” in *Financial Cryptography: 7th International Conference* (R. N. Wright, ed.), no. 2742 in Lecture Notes in Computer Science, pp. 182–189, Springer, 2003.
- [2] M. Blaze, J. Feigenbaum, and J. Lacy, “Decentralised trust management,” in *Proceedings of the 17th Symposium on Security and Privacy*, (Los Alamitos), pp. 164 – 173, IEEE Computer Society Press, 1996.

- [3] Li, Grosf, and Feigenbaum, “A logic-based knowledge representation for authorization with delegation,” in *PCSFW: Proceedings of The 12th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1999.
- [4] B. S. Firozabadi, M. Sergot, and O. Bandmann, “Using Authority Certificates to Create Management Structures,” in *Proceedings of Security Protocols, 9th International Workshop, Cambridge, UK*, pp. 134–145, Springer Verlag, April 2001.
- [5] D. Jaffee, *Organization Theory: Tension and Change*, p. 99. McGraw-Hill, 2001.
- [6] B. S. Firozabadi and M. Sergot, “Power and permission in security systems,” in *Security Protocols* (B. Christianson, B. Crispo, and M. Roe, eds.), no. 1796 in Lecture Notes of Computer Science, (Cambridge, UK), pp. 48–53, Springer Verlag, April 1999.
- [7] D. Povey, “Optimistic security: a new access control paradigm,” in *Proceedings of the 1999 workshop on New security paradigms*, pp. 40–45, ACM Press, 2000.
- [8] D. Povey, “Enforcing well-formed and partially formed transactions for UNIX,” in *Proceedings of the 8th USENIX Security Symposium*, pp. 47–62, 1999.
- [9] G. Stevens and V. Wulf, “A new dimension in access control: studying maintenance engineering across organizational boundaries,” in *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pp. 196–205, ACM Press, 2002.
- [10] T. Jaeger, A. Edwards, and X. Zhang, “Managing access control policies using access control spaces,” in *Proceedings of the seventh ACM symposium on Access control models and technologies*, pp. 3–12, ACM Press, 2002.
- [11] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.
- [12] O. Bandmann, M. Dam, and B. S. Firozabadi, “Constrained Delegations,” in *proceedings of 2002 IEEE Symposium on Security and Privacy*, 2002.