DISCRETIONARY OVERRIDING OF ACCESS CONTROL IN THE PRIVILEGE CALCULUS

Erik Rissanen
SICS
Box 1263
164 29 KISTA
SWEDEN
mirty@sics.se

Babak Sadighi Firozabadi
SICS
Box 1263
164 29 KISTA
SWEDEN
babak@sics.se

Marek Sergot
Department of Computing
Imperial College London
180 Queen’s Gate
London SW7 2BZ
UK
mjs@doc.ic.ac.uk

Abstract We extend a particular access control framework, the Privilege Calculus, with a possibility to override denied access for increased flexibility in hard to define or unanticipated situations. We require the overrides to be audited and approved by appropriate managers. In order to automatically find the authorities who are able to approve an override, we present an algorithm for authority resolution. We are able to calculate from the access control policy who can approve an override without the need for any additional information.

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. There are many reasons why it is difficult to specify the policy completely in advance, and therefore the policy will be incomplete. That will cause a conflict between needs for legitimate access and needs to protect against unauthorised access. We have in a previous position paper Rissanen et al., 2004 categorised access needs as follows:

1 *Anticipated, allowed and machine encodable*: Access 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*: Access 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*: Access 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 formally define "an emergency".)

4 *Anticipated, denied and not machine encodable*: Access 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.

What is needed is some kind of flexibility, which will allow for granting of access rights retroactively. We suggest as a solution to distinguish between what a principal *can* do, what it is *permitted* to do, and what it is *forbidden* to do. The intersection of *can* and *not permitted* is what we refer to as *possibility-with-override* or (sometimes) *ability to override*. In our framework which we present here, the presence of a permission for an access means that the access may be performed. The presence of an possibility-with-override, but no permission for an access means the access may not be performed, but can be performed if the user explicitly overrides the denial. If there is neither a permission or a possibility-with-override, the access is not permitted and cannot be done.

In addition to the possibility to override we introduce the notion of authority resolution, which is an automatic procedure that will, given information about an override and an access control policy, find who is in a position to audit and approve the override.

1.1 Related Work

The idea of being able to override denied access is by no means new. Lee Badger Badger, 1990 describes a formalism for integrity constraints that can be recovered after an override. Many commercial applications, for instance for health care, have emergency override in them. There is also more recent work, which is presented below.

Povey, 2000; Povey, 1999 focus is on guaranteeing system integrity by means of transactions that can be rolled back.

Gunnar Stevens and Volker Wolf Stevens and Wulf, 2002 have performed a case study at a steel mill and found practices to grant access rights either before, during or after an access is performed, which is in line with our ideas.

Jaeger et. al. Jaeger et al., 2002 introduce a concept called *access control spaces*. This concept is used primarily for analysing conflicts in access control policy or to analyse 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 use, 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.

Provisional access control Kudo and Hada, 2000, which is included in XACML OASIS, 2004 in the form of the obligation concept, can be used for instance to specify different access levels and that an access should be logged.

Our main contribution in this paper is the concept of automatic authority resolution, which we have not been able to find any previous work on.

2. Extending the Privilege Calculus

Here we present a framework for decentralised management of authorisations. It is a modified version of the framework presented in Bandmann et al., 2002; Firozabadi et al., 2001, extended to include possibility-with-override. We have chosen this particular framework since it provides information about the source of authorisations.

We want possibility-with-override to be a part of the access control policy, in contrast to a mechanism outside the policy, since for efficiency of implementation and administration it should be manageable in similar ways as regular permissions.

The goals of the original Privilege Calculus were to decentralise access control management and to differentiate between administrative and access level authorisations. All authorisations are expressed in the form of delegation certificates and removals are done by revoking certificates. Administrative rights contained in the certificates dictate which other certificates are considered valid, as explained below.

The Privilege Calculus is based on the concept of "constrained delegation", which means that an administrative right contains constraints on what it applies to and how it may be delegated further. With these constraints it is possible to divide up the management of access control at a central level in the organisation, without the need to micromanage the details. When we developed the override mechanism, our goal was to use this existing division in the access control policy to automatically send notifications of overrides to the right people in the organisation without the need of any central planning specifically for handling of the override audits.

The following presents the semantics of the calculus in a very brief manner. Due to space constraints, for a more thorough understanding, we refer to the original papers.

2.1 Semantics of the Privilege Calculus

Definition 1.. Let $PRIN$ be the set of principals in the system. Further let $\preceq$ denote a *subsumes* relation over $PRIN$ as follows:

- $p \preceq p$ if p is an atomic principal and $p \in \ PRIN$.
- $p \prec P$ if $p \in P$ and $P \subseteq PRIN$;
- $P_1 \preceq P_2$ if $P_2 \subseteq PRIN$, and $P_1 \subseteq P_2$.

Informally the relation $\preceq$ is used for comparing group membership of principals. We assume the existence of the relation, but leave its definition and management outside the scope of this paper. We have chosen to not include groups of objects and actions in order to be brief.

Definition 2.. Let I denote a time interval of type $[t_1, t_2]$, where $t_1, t_2 \in \mathbf{R}$. We define a *subsumes* relation between two time intervals as follows:

- $t_i \preceq [t_1, t_2]$ if $t_1 \leq t_i$ and $t_i \leq t_2$;
- $[t_1, t_2] \preceq [t_3, t_4]$ if $t_3 \leq t_1$ and $t_2 \leq t_4$.

Definition 3.. Let $PRIN$, ACT, and OBJ be (disjoint, non-empty) sets of agents, actions, and objects, respectively. We define the set of privileges Φ inductively as follows:

- *perm*$(s, a, o) : I \in \Phi$, if $s \preceq PRIN, \ a \in ACT,$ and $o \in \ OBJ$;
- *can*$(s, a, o) : I \in \Phi$, if $s \preceq PRIN, \ a \in ACT,$ and $o \in \ OBJ$;

- *auth*$(s, \phi)\,{:}I \in \Phi$, if $s \preceq PRIN$, and $\phi \in \Phi$;
- *auth*$^*(s, \phi)\,{:}I \in \Phi$, if $s \preceq PRIN$, and $\phi \in \Phi$.

I represents the time interval for which a privilege is valid. Privileges of the form *perm*$(s, a, o)\,{:}I$ denote access-level permissions. Privileges of the form *can*$(s, a, o)\,{:}I$ denote access-level possibilities-with-override. Privileges of the form *auth*$(s, \phi)\,{:}I$ and *auth*$^*(s, \phi)\,{:}I$ denote management-level authorities, that is, the right to create the privilege ϕ. The difference between *auth* and *auth** is explained below.

We call s in the above expressions the *subject* of the privilege.

Please note that the privilege expressions themselves do not grant any access rights. Instead they are placed inside authorisation certificates and the validity of the certificates are calculated based on what management-level authorisations are present. Thus, the semantics of these expressions are defined by the following definitions in combination.

Definition 4.. We define the set of declaration certificates Σ^+ and the set of revocation certificates Σ^- as:

- $declares(s, \phi, t, id) \in \Sigma^+$, if $s \in PRIN,\ \phi \in \Phi,\ t \in \mathbf{R}$, and $id \in \mathbf{N}$, where $\mathbf{R}$ denotes the real numbers, and $\mathbf{N}$ denotes the natural numbers;
- $revokes(s, id, t) \in \Sigma^-$, if $s \in PRIN,\ id \in \mathbf{N},$ and $t \in \mathbf{R}$.

Note that declarations and revocations can only be performed by atomic principals and not by groups of principals.

Informally an element $declares(s, \phi, t, id) \in \Sigma^+$ means that s claims at time t that ϕ is true. The definitions below define when such a declaration is considered to be valid.

Definition 5.. We define a comparison relation denoted by $\sqsubseteq$ between two privileges as follows:

$\phi \sqsubseteq \psi$ if:

1 $\phi =$ *perm*$(s_1, a, o)\,{:}I_1$, $\psi =$ *perm*$(s_2, a, o)\,{:}I_2$, $s_1 \preceq s_2$ and $I_1 \preceq I_2$;

2 $\phi =$ *can*$(s_1, a, o)\,{:}I_1$, $\psi =$ *perm*$(s_2, a, o)\,{:}I_2$, $s_1 \preceq s_2$ and $I_1 \preceq I_2$;

3 $\phi =$ *can*$(s_1, a, o)\,{:}I_1$, $\psi =$ *can*$(s_2, a, o)\,{:}I_2$, $s_1 \preceq s_2$ and $I_1 \preceq I_2$;

4 $\phi =$ *auth*$(s_1, \alpha)\,{:}I_1$, $\psi =$ *auth*$(s_2, \beta)\,{:}I_2$, $\alpha \sqsubseteq \beta$, $s_1 \preceq s_2$ and $I_1 \preceq I_2$;

5 $\phi =$ *auth*$(s_1, \alpha)\,{:}I_1$, $\psi =$ *auth*$^*(s_2, \beta)\,{:}I_2$, $\alpha \sqsubseteq \beta$, $s_1 \preceq s_2$ and $I_1 \preceq I_2$;

6 $\phi = \textit{auth}^*(s_1, \alpha):I_1$, $\psi = \textit{auth}^*(s_2, \beta):I_2$, $\alpha \sqsubseteq \beta$, $s_1 \preceq s_2$ and $I_1 \preceq I_2$;

7 $\psi = \textit{auth}^*(s_1, \beta):I_2$, $\phi \sqsubseteq \beta$

8 $\phi = \textit{auth}(s_1, \alpha):I_1, \psi = \textit{auth}^*(s_2, \beta):I_2$ if $s_1 \preceq s_2, \alpha \sqsubseteq \textit{auth}^*(s_2, \beta):I_2$, and $I_1 \preceq I_2$;

9 $\phi = \textit{auth}^*(s_1, \alpha):I_1, \psi = \textit{auth}^*(s_2, \beta):I_2$, if $s_1 \preceq s_2, \alpha \sqsubseteq \textit{auth}^*(s_2, \beta):I_2$, and $I_1 \preceq I_2$.

These comparisons are used below to make sure that administrators do not exceed their authorisations when delegating. We can see in 2 that a permission implies a possibility-with-override, that is, if an administrator can create a permission, he will also be able to create the weaker privilege of possibility-with-override for the same object and action.

The $\textit{auth}^*()$ construct needs some explanation. It is used to give flexibility for administrators. The authorisation $\textit{auth}(p, \textit{auth}(G, perm(G, o, a):I_1):I_2):I_3$ means that we permit p to appoint an administrator from within the group G, who then in turn can create access permissions for object o and action a for principals within group G. Let us call this administrator g. p will be limited in that he must appoint an administrator from G and will not be able to issue the access level permission himself. Also, p cannot create more than one immediate administrator, that is p will hand the right to g, who in turn will create the permission to access o. If we instead create the authorisation $\textit{auth}(p, \textit{auth}^*(G, perm(G, o, a):I_1):I_2):I_3$, we will give additional possibilities to p. p will be able to create the access level permission directly if he chooses to do so, as given by rule 7. He can appoint an administrator g as previously, as given by rule 8. He can also permit g to delegate the authority in several steps by appointing intermediary managers chosen from G. This allows p to let subordinates organise their own sub-organisations within G. For instance we could have p delegate to g who will delegate to g' who will in turn create the access level permission. The use of the $\textit{auth}^*()$ construct is explained in more detail in Bandmann et al., 2002.

Definition 6.. We define a certificate database to be a tuple $\mathcal{D} = (\mathbf{SoA}, \mathbf{D}^+, \mathbf{D}^-)$, where $\mathbf{SoA} \subset \Phi$ is a finite set of *Source of Authority* privileges, $\mathbf{D}^+ \subset \Sigma^+$ is a finite set of declaration certificates and $\mathbf{D}^- \subset \Sigma^-$ is a finite set of revocation certificates. It is the combined contents of this certificate database that will decide which accesses are permitted.

We adopt the following constraints on a certificate database.

1 If $declares(s_1, \phi_1, t_1, id) \in \mathbf{D}^+$, and $declares(s_2, \phi_2, t_2, id) \in \mathbf{D}^+$, then $s_1 = s_2$, $\phi_1 = \phi_2$, and $t_1 = t_2$. This says that $\mathbf{D}^+$ cannot contain two different certificates with the same id.

2 If $declares(s_1, \phi, t_1, id) \in \mathbf{D}^+$ and $revokes(s_2, id, t_2) \in \mathbf{D}^-$, then $s_1 = s_2$ and $t_1 \leq t_2$. This says that a certificate can be revoked only by its issuer and not before it is declared. In fact, the first restriction can be relaxed but this introduces the need for extra components which are omitted here for simplicity.

3 If $revokes(s_1, id, t_1) \in \mathbf{D}^-$ and $revokes(s_2, id, t_2) \in \mathbf{D}^-$, then $s_1 = s_2$ and $t_1 = t_2$. This says that there cannot be two revocations of the same declaration certificate in the same database. We adopt this restriction to simplify the database in order to streamline the theory.

Definition 7.. Let $\vdash$ be the *validates relation* between a privilege and a declaration certificate, where

- *auth*$(s_2, \phi_2) : I \vdash declares(s_1, \phi_1, t, id)$, if $s_1 \preceq s_2$, $\phi_1 \sqsubseteq \phi_2$ and $t \preceq I$;

and,

- $\Gamma \vdash d,$ if $\Gamma \subseteq \Phi,$ and $\exists\, q \in \Gamma$ such that $q \vdash d.$

Informally this defines the semantics of the administrative permission *auth*$()$, that is, it makes us consider certain declarations to be valid. Also notice that the *auth*$^*()$ form does not validate a declaration. *auth*$^*()$ is only used inside an *auth*$()$ expression. We will use the validates relation below to recursively define which permissions are valid.

Definition 8.. We define the set of *effective* declaration certificates $\mathbf{E}_{\mathcal{D}}(t) \subseteq \mathbf{D}^+$ of a database $\mathcal{D}$ at a certain time t, as:

$$\mathbf{E}_{\mathcal{D}}(t) = \{declares(s, p : I, t_1, id) \in \mathbf{D}^+ |\; t \preceq I \wedge \\ revokes(s, id, t_2) \in \mathbf{D}^- \rightarrow t_2 > t\}.$$

Informally, we define that the interval I defines when the authorisation is usable.

Definition 9.. Let $d_1, d_2 \in \mathbf{D}^+$, where $d_1 = declares(s_1, \phi_1, t_1, id_1)$ and $d_2 = declares(s_2, \phi_2, t_2, id_2)$. We define the *supports* relation $S_{\mathcal{D}}$ as follows:

$$d_1 \; S_{\mathcal{D}} \; d_2 \text{ if } d_1 \in \mathbf{E}_{\mathcal{D}}(t_2),\; \phi_1 \vdash d_2 \text{ and } t_1 < t_2.$$

Informally, we define that a declaration depends on another previous declaration to be valid.

We have modified this definition compared with the presentation in Firozabadi et al., 2001 in that we have added the condition $t_1 < t_2$ to prevent cycles in the support relation. Cycles are not possible in Firozabadi et al., 2001, but together with the *auth** form from Bandmann et al., 2002 cycles become possible unless prevented with this extra constraint. Although cycles are not a problem in principle, the authority resolution algorithm later on becomes more complicated to explain for a cyclic graph, so we make this simplification.

Definition 10.. The set of certificate chains $C_{\mathcal{D}}$ in a certificate database $\mathcal{D}$ is the transitive closure of $S_{\mathcal{D}}$.

Definition 11.. We define the set of true privilege statements at a time-point t, in our calculus, by defining function $h_{\mathcal{D}} : \mathbf{R} \rightarrow 2^{\Phi}$ as:

$$\begin{aligned} h_{\mathcal{D}}(t) = \{p \mid\ & p\!:\!I \in \Phi \wedge \\ & (p\!:\!I \in \mathbf{SoA} \vee \\ & (d_1, declares(s, p\!:\!I, t_2, id)) \in \mathbf{C}_{\mathcal{D}} \wedge declares(s, p\!:\!I, t_2, id) \in \mathbf{E}_{\mathcal{D}}(t) \wedge \\ & \mathbf{SoA} \vdash d_1)\}. \end{aligned}$$

We also say that a privilege p *holds* at time-point t when $p \in h_{\mathcal{D}}(t)$.

Informally, this means that although anybody can make a privilege statement in the form of a declaration certificate, we will not accept the statements as true unless they can be traced back to the SoA.

2.2 Access Requests

When we receive an access request, which is a tuple in the form of (u, o, a, t), where u is a principal, o an object, a an action and t is the time of access, we search among $h_D(t)$ for a $perm(s, o, a)\!:\!I$ such that $u \preceq s$ and $t \preceq I$. If there is such a permission, then the response is “yes”. In case there is no permission, we search for a $can(s, o, a)\!:\!I$ such that $u \preceq s$ and $t \preceq I$. If there is such an ability then the response is ‘requires override’. In that case the user would be presented with the option to override the denied access and the application will log the access if the users chooses to override. If there is neither a matching permission nor a possibility-with-override, the response is ‘access denied’.

3. Approval mechanism and authority resolution

When a user performs an override to make an access, the override is logged, and a message is sent to an appropriate authority for approval. In a large organisation it may not be possible to have a single person or unit which is able to comprehend or have authority over the whole organisation. We therefore need to decentralise the responsibility of audit and approval of overrides. We call the search for an appropriate authority for a given override *authority resolution*.

3.1 Approval Mechanism Properties

In our earlier paper we identified two properties for an authority resolution mechanism. The mechanism should be:

- *Safe:* Only legitimate authorities should be notified.
- *Unobtrusive:* Among the legitimate authorities, we should notify those who are most likely to understand the override and least likely to be bothered unnecessarily.

The first property is critical, but it is easily defined, as we will show below. The second property is not critical if we define the approval mechanism appropriately and all legitimate authorities are potentially consulted. In that case the ordering is thus somewhat arbitrary.

Since an approval of an override is in effect a retroactive granting of a permission, the authorities who should be able to approve an override are precisely those who can create a permission for the access that was overridden. In this framework they correspond to the subjects of effective certificates who have a valid support chain from the SoA such that their certificates support the creation of a permission for the access at the time of the override. So, for an access override (u, o, a, t) that is approved at time t', they are all authorisations from $h_D(t') \cap E_D(t')$ of the form *auth*$(s_1, perm(s_2, o, a) : I_2) : I_1)$ such that $u \preceq s_2$, $t \preceq I_2$ and $t' \preceq I_1$. In this case s_1 would be a legitimate authority.

Since it is possible that there are multiple legitimate authorities for approving a given override, we would like to contact them in such an order that we are least likely to bother many authorities.

We note that the access control framework we are using does not contain negative permissions. We do not wish to introduce negative permissions just because of the override approval mechanism. Since we view an approval as a retroactive granting of a permission, in case some authorities approve and some disapprove, the approvals should have precedence. If all of them disapprove (or do not care), we view the override as disapproved. With these semantics we can define an approval mechanism in which the order of authorities notified does not affect the result.

For ordering the authorities 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. The source of authority of a resource is always a legitimate authority, but we want to keep him last in the notification list since he is the highest authority. For authorities between the SoA and the lowest level administrators, we can use the order of their appearance in the chain as a heuristic. In case of parallel chains we can use an arbitrary ordering or notification in parallel.

3.2 An Algorithm for Authority Resolution

Here we present a simple algorithm that is based on the above discussion. Input to the algorithm is a performed override and a certificate database.

The algorithm consists of two parts. In the first part we create a reduced graph from the delegation database. Let G_d be the graph that describes the delegation database by letting there be a node in G_d for each certificate and an edge between nodes that correspond to certificates between which there is a direct support relation. G_d is directed and acyclic. It is acyclic because of the condition on the time stamps in definition 9 of the access control framework.

Now form the reduced graph G_r by letting there be a node in G_r for each certificate from $h_D(t) \cap E_D(t)$ which authorises approval of the given override (as explained earlier). Let there be an edge in G_r between two nodes if there is a path between the corresponding nodes in G_d. The motivation behind this is that we want to remove all certificates that do not empower approval (to satisfy the safety property of the authority resolution), but still keep as much of the structure of decentralisation as possible (to be able to satisfy the unobtrusiveness property).

G_r can be calculated by performing a depth first search on G_d starting only from the nodes that will be in G_r. When doing the search we need to keep in each node, n, a lists of nodes, which will be filled with a list of all nodes of G_r which can be reached from n. Once the search is complete, these lists will give the edges of G_r.

In the second part of the algorithm we order the authorities by means of a modified breadth first search on G_r from the bottom going up.

```
$R \leftarrow$ empty list of sets of principal names
for each node $n$ of $G_r$
    $n.counter \leftarrow$ number of children of $n$
$S \leftarrow$ the set nodes for which $counter = 0$
do while $S$ is not empty
    add the set of subjects of all nodes in $S$ to $R$
    $Q \leftarrow S$
    $S \leftarrow \emptyset$
    for each $q$ in $Q$
        for each parent $p$ of $q$
            reduce $p.counter$ with one
            if $p.counter$ is zero
                add $p$ to $S$
```

Table 1 lists some sample certificates. Figure 1 shows the support relations among those certificates and illustrates the first part of the algorithm.

Id	*Issuer*	*Authorisation*
1	r	$\mathbf{\mathit{auth}}(b, \mathbf{\mathit{auth}}^*(G, perm(G, o, a)))$
2	b	$\mathbf{\mathit{auth}}(c, \mathbf{\mathit{auth}}(G, perm(G, o, a)))$
3	c	$\mathbf{\mathit{auth}}(d, perm(G, o, a))$
4	d	$can(e, o, a)$
5	b	$\mathbf{\mathit{auth}}(f, \mathbf{\mathit{auth}}^*(G, perm(G, o, a)))$
6	f	$\mathbf{\mathit{auth}}(g, \mathbf{\mathit{auth}}^*(G, perm(G, o, a)))$
7	g	$\mathbf{\mathit{auth}}(h, \mathbf{\mathit{auth}}^*(G, perm(G, o, a)))$
8	f	$\mathbf{\mathit{auth}}(h, \mathbf{\mathit{auth}}^*(G, perm(G, o, a)))$
9	h	$\mathbf{\mathit{auth}}(i, perm(G, o, a))$
10	i	$can(e, o, a)$

Table 1 Example delegation certificates. For brevity we have not included the time intervals. The validity intervals of all the authorisations are [1,100] and all of the certificates are issued at the time point equal to the id of the certificate.

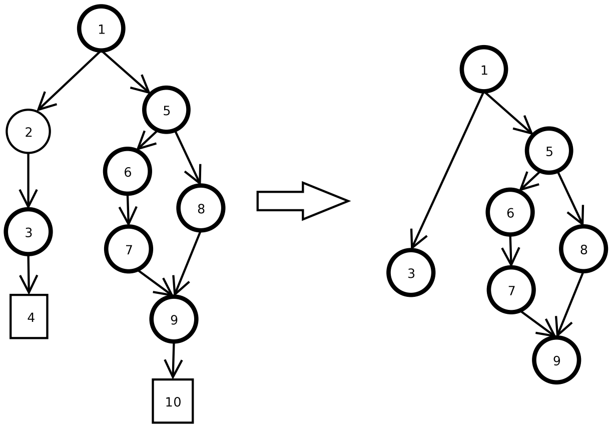

Figure 1. Example of the graph reduction step in the authority resolution algorithm. The graph on the left shows the support relations between the certificates in table 1. Circles represent certificates that grant administrative authorisations. The rectangles represent certificates that grant abilities. Thick circles are certificates that grant authority to issue a permission. The graph to the right shows the reduced graph. Certificate 2 does not support direct granting of access rights. Certificates 4 and 10 do not represent administrative permissions. The remaining certificates all permit the granting of access permissions, thus approval.

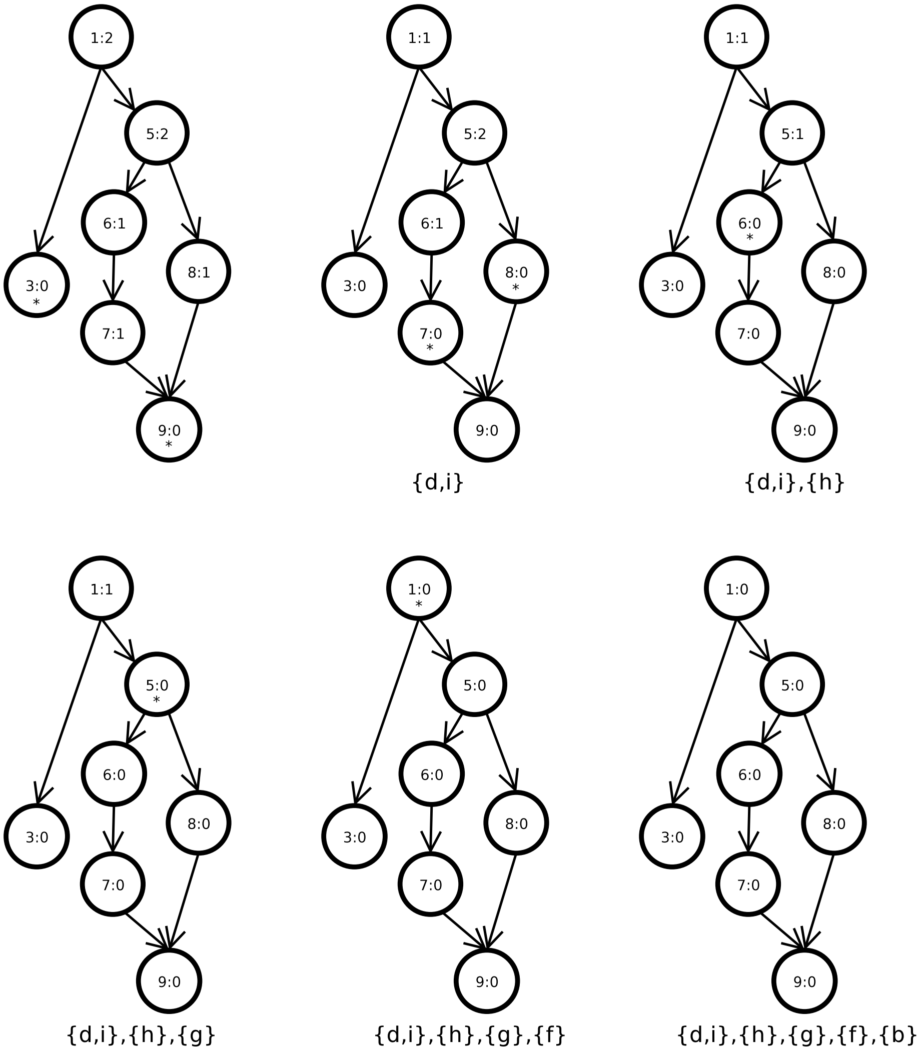

Figure 2. Example of the authority resolution algorithm. The algorithm works on the reduced graph from figure 2. Each figure presents the state of the algorithm at each iteration of line 5. The asterisks represent the set S. The numbers after the colons inside the nodes are the counters. The lists below the graphs are the accumulated result lists of the algorithm. The final list is the output of the algorithm. In this case we should notify the users d and i first, and if neither approve, notify h, g, f, and b in that order. If none of them approves, the override is considered unauthorised.

Figure 2 illustrates the second part of the algorithm.

The result is an ordered list of sets of authorities to notify. We would send the notification to all authorities in the first set. As a special case we could divide this set into those who have issued a relevant possibility-with-override, and notify them before the others.

In case someone approves the override we do not notify anyone else and the override is considered to be approved. In case all notified authorities either disapprove or take no action, we would notify the next set of authorities from the list, and so on. If anyone approves, the override is considered to be approved. If in the end no one has approved the override, we view the override as disapproved, and the relevant authorities can take some kind of sanctioning action. We leave the coordination of the sanctioning outside the scope of this paper. We can see that the order in which authorities are notified does not affect the end result.

Properties of the Algorithm. To see that the algorithm terminates, we note that because a counter is reduced just before it is tested at line 12, a node can be included in S only once. Because of line 8, S will be cleared in every iteration of the loop. Since the number of nodes is finite, eventually there will be no more nodes which can be included in S at line 13, and the loop at line 5 will terminate.

Define the *upper height* of a node N as the length of the longest path originating from N. We can prove by induction on the upper height that every node of the graph will be included in S.

Theorem:. A node with upper length n will be included in S.
Proof: Any node with upper height 0 will be included in S in lines 1-4. Thus the theorem is true for $n = 0$.
Now, assume that the theorem is true for $n = k$. If a node N has upper height $k + 1$, then all its children must have upper height k or less. By our assumption, all those children will be included in S before the algorithm terminates. Then, because of lines 9-13, the counter of N will reach zero and N will be included as well. Thus we have proved that a node with upper height $k+1$ will be included in S, and by induction it follows that any node with upper height of 0 or more will be included in S. ■

Since every subject of the nodes of the graph will be included in the result, all possible authorities will be included in the result.

We choose to not formalise the order in which the notifications are generated, but just note that since we start from the bottom, lower level managers will be notified before higher level managers.

4. Conclusion and Further Work

In many cases it is not possible to define the security policy completely in a machine readable form and we may not anticipate all needs. The incompleteness of the policy will lead to a conflict between need to protect against unauthorised access and the need for legitimate access. In case availability is important, a solution may be to allow users to override access denials and then have managers audit the override. Authority resolution is a mechanism for improving the efficiency of this audit. We have shown that it is possible to use only existing information in the Privilege Calculus framework to implement authority resolution.

What we present is early work and many issues remain. Of main interest is to perform a study of the usefulness of the approach.

Another area is to improve on the work-flow of the mechanism, which right now is not as good as we wish in the case of disapproval of override. By using an access control framework with negative permissions or additional information, different mechanisms for propagation of notifications may be possible.

We are also interested in applying our ideas to other access control frameworks such as XACML.

References

Badger, Lee (1990). Providing a flexible security override for trusted systems. In *Computer Security Foundations Workshop III, 1990. Proceedings*, pages 115–121.

Bandmann, Olav, Dam, Mads, and Firozabadi, B. Sadighi (2002). Constrained Delegations. In *proceedings of 2002 IEEE Symposium on Security and Privacy*.

Firozabadi, B. Sadighi, Sergot, M., and Bandmann, O. (2001). Using Authority Certificates to Create Management Structures. In *Proceedings of Security Protocols, 9th International Workshop, Cambridge, UK*, pages 134–145. Springer Verlag.

Jaeger, Trent, Edwards, Antony, and Zhang, Xiaolan (2002). Managing access control policies using access control spaces. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 3–12. ACM Press.

Kudo, Michiharu and Hada, Satoshi (2000). Xml document security based on provisional authorization. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 87–96. ACM Press.

OASIS (2004). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

Povey, Dean (1999). Enforcing well-formed and partially formed transactions for UNIX. In *Proceedings of the 8th USENIX Security Symposium*, pages 47–62.

Povey, Dean (2000). Optimistic security: a new access control paradigm. In *Proceedings of the 1999 workshop on New security paradigms*, pages 40–45. ACM Press.

Rissanen, Erik, Firozabadi, Babak Sadighi, and Sergot, Marek (2004). Towards a mechanism for discretionary overriding of access control, position paper. Presented at Security Protocols, 12th International Workshop, Cambridge, UK.

Stevens, Gunnar and Wulf, Volker (2002). A new dimension in access control: studying maintenance engineering across organizational boundaries. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 196–205. ACM Press.