Comparing and Integrating Break-the-Glass and Delegation in Role-based Access Control for Healthcare

Ana Ferreira[1] and Gabriele Lenzini[2]
[1]*CINTESIS - Centre for Health Technologies and Services Research, Faculty of Medicine, University of Porto, Rua Dr. Plácido da Costa, 4200-450 Porto, Portugal*
[2]*SnT - Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, 4 rue A. Weicker, L-2127 Luxembourg, Luxembourg*

Keywords: RBAC in Healthcare, Break The Glass, Delegation, Access Control.

Abstract: In healthcare security, Role-based Access Control (RBAC) should be flexible and include capabilities such as Break-the-Glass and Delegation. The former is useful in emergencies to overcome otherwise a denial of access, the latter to transfer rights temporarily, for example, to substitute doctors. Current research studies these policies separately, but it is unclear whether they are different and independent capabilities. Motivated to look into this matter, we present a formal characterization of Break-the-Glass and Delegation in the RBAC model and we inquire on how these two policies relate. After giving arguments in favour of keeping them apart as different policies, we propose an RBAC model that includes them.

1 INTRODUCTION

The RBAC model (Sandhu et al., 1996) is a widespread standard for the administration of accesses in organizations with a high number of employees each having specific roles. Access control policies are defined per role instead of per simple employee. In healthcare, RBAC has been customized to handle situations where doctors find themselves requesting access to data that they should not normally access (Ferreira et al., 2007); in such situations a role must be given allowances that the standard policies would deny. This may happen when a doctor intervening in life-saving emergencies, substituting a colleague, or consulted for a second opinion needs to access the records of another doctor's patient. To work with emergences RBAC has been extended with a feature called *Break-the-Glass (BTG)* (Ferreira et al., 2006): it allows a role to overrule the restrictions that normally constrain the access to data but the violation is recorded and the violator usually bound to fulfil specific obligations (Ferreira et al., 2009). To work with referrals, RBAC has been extended with an operation called *Delegation (DELG)* (Barka and Sandhu, 2000; Wang and Osborn, 2006; Barka and Sandhu, 2007): it enables subjects to transfer permissions (potentially all the permissions linked to a role) to other subjects.

BTG and DELG have already been studied in the context of RBAC but the works that consider one do not study the other (Barka and Sandhu, 2000; Wang and Osborn, 2006; Barka and Sandhu, 2007; Crampton and Khambhammettu, 2008; Ferreira et al., 2009). The few works that address both i.e., (Crampton and Morisset, 2011; Krautsevich et al., 2012) are not motivated by comparing the two capabilities but rather by showing how to implement break-the-glass as auto-delegation.

Understanding whether BTG and DELG can co-exist as independent capabilities is instead what motivates the work we report on this paper. To shed light on the matter we need to compare the definitions and implementations of these two policies and reason on whether and how they relate. Thus we study the RBAC models that express those policies separately; after giving an argument in favour of the co-existence of the two policies, we propose a RBAC model that combines them and study the security implications that this combination has. To our knowledge such research is unprecedented.

Contribution. This paper describes the RBAC model in preparation to discuss and compare BTG and DELG. The paper's approach is to reason on the syntax and on the operational semantics of the policies in terms of the procedure that checks whether a request is authorized (§3). In the specific context of healthcare access control, this paper studies the *re-*

Ferreira, A. and Lenzini, G.
Comparing and Integrating Break-the-Glass and Delegation in Role-based Access Control for Healthcare.
DOI: 10.5220/0005683600630073
In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016)*, pages 63-73
ISBN: 978-989-758-167-0

quirements that make a set of policies secure. Secure here means free from ambiguities that would let one user gain a permission without breaking-the-glass on it and without receiving legitimately the permission by delegation (§4–§5): due to the nature of BTG and DELG, the first permitting controlled exception and the second spreading permissions, identifying when violations can happen is not obvious.

This paper questions whether BTG and DELG can be distinct or not. At least in the medical domain where managing and assigning permissions is decentralized, it turns out that there is an argument for keeping the two policies apart, which leads to more expressive security policies. This is a novel argument with respect to the state of the art (§2). Thus, this paper proposes a new RBAC model that supports policies with interleaved BTG and DELG (§6) and shows, via examples, that this gives flexibility in writing access control policies for coping with the heterogeneous affairs of the daily practise in healthcare. This paper also identifies two relevant security requirements for this new model: policy makers can use them to write security-by-design policies.

2 RELATED WORK

This work relates with previous research on access control models for delegation and break-the-glass.

Delegation has been widely studied in the context of RBAC models. The majority of the works define the capabilities of this operation (Wainer, 2005; Wang and Osborn, 2006; Barka and Sandhu, 2007; Crampton and Khambhammettu, 2008; Li and Wang, 2008; Hasebe and Mabuchi, 2010) but only a few of them trial and test delegation in real healthcare scenarios.

Research on break-the-glass spans from the definition of concepts (Rissanen et al., 2006; Rostad and Edsberg, 2006; Brucker and Petritsch, 2009) to the implementation of this feature in a hospital system (Ferreira et al., 2009).

With respect to these works, we discuss break-the-glass and delegation with the aim of understanding whether they can be combined and integrated. Understanding the relation between break-the-glass and delegation is a research goal not addressed previously. The few works that discuss break-the-glass and delegation with RBAC intend to show how to implement break-the-glass as auto-delegation on a permission (Crampton and Morisset, 2011). Auto-delegation is authorized only when no other role that possesses legitimately the permission is present in the system, and resolving whether a role is authorized to auto-delegate a permission may become conditional to the probability that such a role is available (Krautsevich et al., 2012). Both works do not argue (it is not in their research's scope) whether there are interpretations of one capability that cannot be captured by the other; consequently, they do not study, as we do, the consequences of using them together within the same RBAC model.

We are motivated to clarify exactly this relation and to discuss the security implications of having break-the-glass and delegation coexisting in the same framework. Our intuition, an argument that we develop in this paper, is that break-the-glass —more apt to cope with not necessarily unforeseen but however exceptional situations— and delegation —suitable to plan transfer of rights— are different features (thus different policies) that call for different security requirements.

That there may be a distinction between delegation and break-the-glass seems emerging from existing implementations of the Break-the-Glass Role-based Access Control (BTG-RBAC) model (Ferreira et al., 2009). Here, breaking-the-glass on a permission mimics what happens when one breaks a protection glass. Glass-protected permissions become and remain accessible until the glass is repaired (i.e., the protection-status is reset). Users with the same role as the one who broke the glass can access the resource without obligations: obligations will come again when the glass is restored. Allowing roles to take advantage of a broken glass without obligations seems to invite for abuses, which we intend to avoid, but even if we assume an implementation that mimics less faithfully what happens to a real glass (see § 4) Ferreira *et al.*'s proposal of the break-the-glass diverges from auto-delegation.

There are still other works that define ways to extend BTG-RBAC, but their applicability is limited to specific domains such as Wireless Sensor Networks (Maw et al., 2014) and Cloud Computing (Rajesh and Nayak, 2012). There is no understanding about how break-the-glass and delegation relate in general and in the medical domain in particular.

3 RBAC AND ITS EXTENSIONS

This section recalls the basic RBAC models in a common and simple formalism with the goal to introduce the approach followed throughout the paper: define the policy language and its operational semantics by specifying how the *authorization procedure* —the procedure that checks whether a request from a user should be allowed or denied— works. In the same formalism, this section describes RBAC with Break-

the-Glass and RBAC with Delegation, together with a specific healthcare use-case of their application in practice.

This allows us to discuss and compare two relevant *security requirements* for Break-the-Glass and Delegation and to discuss whether and how the policies can coexist without introducing vulnerabilities in the security outfit that the policies are meant to supply for the system.

Assumptions. We assume that there is no centralized management by a third-party's administration: all operations are performed by users i.e., doctors, patients or their delegates. This is because in the healthcare domain, users need to be independent in using the system in order to cope with sudden and dynamic situations. A third-party would introduce one more step in the process, potentially compromising effectiveness and efficiency.

We assume policies are set at start-up, e.g., by the owner or the manager of the data. This *initial policy set* imprints the system's security.

We assume that break-the-glass and delegation are applied on permissions and not on roles. This fine-grained option fits better to the variety of situations often decided on a need-to-know basis that characterize the daily accesses by the healthcare professionals.

3.1 Permissions, Operations, Objects

A *permission* tells what *operation* is permitted on what *object*. Let *OPS* be the set of operations and *OBS* the set of objects; the set of permissions is $PRMS \subseteq OPS \times OBS$. Each $p \in PRMS$ has the form (op, obj), where $op \in OPS$ and $obj \in OBS$. Having permission (op, obj) (e.g., $(\texttt{read}, \texttt{f})$) means be allowed to execute operation *op* (i.e., `read`) on object *obj* (i.e., `f`), or equivalently said, to be allowed to execute $op(obj)$ (i.e., $\texttt{read}(\texttt{f})$).

3.2 RBAC Basic Models

The simplest RBAC models are called RBAC_0 and RBAC_1 (Sandhu et al., 1996).

RBAC_0 consists of several sets: *SESSIONS* of session, *ROLES* of roles, *USERS* of users, and *PRMS* of *basic* permissions (we qualified permissions in *PRMS* as "basic" to distinguish delegation and break-the-glass, which we introduce later). RBAC_0 comprises the following relations: $UA \subseteq USERS \times ROLES$, the roles that users can play; $PA \subseteq PRMS \times ROLES$, the permissions roles have. Set $R_u = \{r : (u, r) \in UA\}$ specifies the roles that a user *u* can play.

RBAC_1 is slightly more elaborated than RBAC_0. It comes with a closed-under-transitivity partial order $\leq$ on $ROLES \times ROLES$ that defines a hierarchy on roles. Set $\downarrow(R_u) = \{r \in ROLES : r \leq r', r' \in R_u\}$ is the set of all roles and all sub-roles that user *u* can play. Within one session, a user *u* can only activate one of its roles, $r_u \in R_u$. Set $\downarrow(r_u) = \{r \in ROLES : r \leq r_u\}$ defines *u*'s role and sub-roles according to the hierarchy. All users and their active permissions in a session are stored in a *policy set* $\pi \subseteq USERS \times PRMS$, $\pi = \{(u, p) \mid u \in USERS,\ r \in \downarrow(r_u), (p, r) \in PA\}$. To determine whether user *u* has permission *p* is simply a matter of checking π on whether *p* is assigned to one of the roles that *u* can play. We let the predicate $Auth_\pi(u, p)$ do this job: it evaluates false (`ff`) for 'permission denied', and true (`tt`) for 'permission allowed'.

$$Auth_\pi : USERS \times PRMS \to \{\texttt{ff}, \texttt{tt}\}$$

$$Auth_\pi(u, p) \stackrel{\text{def}}{=} \begin{cases} \texttt{tt}, & \text{if } (u, p) \in \pi \\ \texttt{ff}, & \text{otherwise} \end{cases}$$

Obligations. These are requests for actions triggered when a user tries to access a resource (Zhao et al., 2007). To handle obligations we need to slightly change *PRMS* to $OPRMS = PRMS \times 2^{OBGS}$ as well as $Auth_\pi$ to make it return a set $\mathbf{o}$ of obligations:

$$Auth_\pi(u, p) \stackrel{\text{def}}{=} \begin{cases} (\texttt{tt}, \mathbf{o}) & \text{if } (u, p) \in \pi \\ (\texttt{ff}, \mathbf{o}), & \text{otherwise} \end{cases}$$

To simplify the notation, in the remainder we omit obligations, assuming that they can be introduced any time, with no theoretical difficulty.

3.3 Running Example (Prologue)

We motivate the use of BTG and DELG in a running example. We consider a situation that is described in the ISO standard (ISO/TS, 2009). The access control is exercised with roles, which include healthcare professionals and the patient or the patient's guardian. The protected resource is the Electronic Health Records (EHR). EHR are used to keep track of patients' medical history, together with lab tests and demographics. EHR can be shared by different institutions and allow for both decentralised and centralised access to the whole data base or part of it by healthcare professionals and patients.

Our use case has four roles (see Table 1): Dr. John, the doctor; Rachel, Dr. John's patient; Michel, Dr. John's assistant; and Dr. Mario, a colleague that occasionally substitutes for Dr. John. Rachel has received the result of her blood tests and she is positive to HIV, a worrisome outcome. Although she knows that false positives can happen,

Table 1: Actors in our use case.

Dr. John	The doctor
Rachel	Dr. John's patient
Michel	Dr. John's assistant
Dr. Mario	Dr. John's substitute

she hastily tries to contact Dr. John to discuss the next steps. But Dr. John is travelling and cannot be reached.

Michel however has been given permission to put Dr. John's patients in contact with Dr. Mario. When that happens Dr. Mario needs access to Dr. John patient's EHR (in this case to Rachel's blood test) and retain that right at most until Dr. John's return.

This situation can be handled in various ways. Dr. John can entrust Dr. Mario directly and delegate him as soon as he leaves. Or he can entrust Michel, and let her have the right to delegate the permission to Dr. Mario when that comes into need. These and other solutions can be implemented using delegation, break the glass, or both. For instance, Dr. John can entrust Michel, to break-the-glass on transferring the right to read Rachel's data to Dr. Mario. The use of BTG here, is because Dr. John wants neither Michel to have the right to read herself nor Dr. Mario to gain the right before time; moreover, he intends the delegation to Dr. Mario be happening only when patients cannot really wait for his return. Dr. John knows that obligations coming with the BTG will restrain Michel from abusing that right.

4 BREAK-THE-GLASS (BTG)

BTG allows to overrule the enforcement mechanism which would normally deny access. The violation must happen justifiably and controllably, so BTG has been integrated into RBAC with obligations: breaking has consequences, for instance, alerting a responsible party.

We write a BTG's permission as $BTG(p)$, where $p = (op, obj)$ is also a permission. So $BTG(p)$ stands for $(\texttt{btg}.op, obj)$, where $\texttt{btg}.op$ is intended to be the operation "break-the-glass on *op*". For instance, `btg.read` is "break-the-glass on read".

All RBAC's sets are consequently extended. For instance, the permission set *PRMS* becomes $PRMS^B = PRMS \cup \{BTG(p) : p \in PRMS\}$. We define $PRMS^B$ in such a way that it does not contain nested BTG policies e.g., $BTG(BTG(p))$. We exclude these policies as not useful: they suggest multiple "glasses" protecting a resource, so distorting the meaning and delaying the BTG action. Relation *PA* becomes $PA^B \subseteq PRMS^B \times ROLES$. The policy set, which we still write π instead of π^B to simplify the notion, is also changed accordingly.

These definitions look trivial, but the way in which BTG works is peculiar. First, a user asks for authorization on $p \in PRMS$, not on $BTG(p)$. Then, if he is not authorized to p but is authorized to $BTG(p)$, he can break the glass; but he has to give consent, aware that there will be consequences (Ferreira et al., 2009).

We show how this mechanism works but in a way that differs from that proposed by Ferreira *et al.* (Ferreira et al., 2009). There is no BTG-state to remind that a glass has been broken, and no next "requesters-by" can take advantage of a glass already broken.

Function $Auth^B_\pi : USERS \times PRMS \to \{\texttt{ff}, \texttt{tt}\}$ behaves exactly as $Auth_\pi$, and accepts a request for a permission that is not a BTG. The auxiliary procedure $Auth^{B'}_\pi$, which we report below, represents the core decision making in the $Auth^B_\pi$. The user's decision of the break-the-glass is collected somewhere else, and here is given as an input parameter, a, which ranges on 'y' = "*I want to break the glass*" and 'n' = "*I do not want to break the glass*".

$$Auth^{B'}_\pi : USERS \times PRMS \times \{\text{y}, \text{n}\} \to \{\texttt{ff}, \texttt{tt}\}$$

$$Auth^{B'}_\pi(u, p, a) \stackrel{\text{def}}{=} \begin{cases} \texttt{tt}, \text{if } Auth^B_\pi(u, p); \\ \texttt{ff}, \text{if } \neg Auth^B_\pi(u, p) \wedge (a = \text{n}); \\ (u, (BTG(p))) \in \pi \\ \text{if } \neg Auth^B_\pi(u, p) \wedge (a = \text{y}); \end{cases} \quad (1)$$

Informally u is authorized either when plays a role that has p or when, despite not having p, has a role that has $BTG(p)$ and his answer is y.

Security Requirements. We have identified only one security requirement for π. Advocating the viewpoint of (Ferreira et al., 2009), break-the-glass is reserved for exceptional situations and must not be possessed by default: so, having permission to BTG must be stated in π. This requirement together with how (1) works suggest that having p makes having $BTG(p)$ superfluous. But since from a security standpoint no harm is done (simply, the third branch in (1) is never executed) this last is more a consistency requirement than a security one.

4.1 Running Example (Continued)

If we want to allow the situation illustrated in §3.3 using BTG only, we have to authorize Dr. Mario to break the glass on reading. This must be set in π according to our requirement. For readability, we write $op(obj)$ instead of (op, obj).

$$\pi = \{(\texttt{DrJohn}, (\texttt{read}(\texttt{blood\_test}))); \\ (\texttt{DrMario}, (\texttt{btg}(\texttt{read}(\texttt{blood\_test}))))\}$$

This setting is however unsatisfactory. Dr. Mario has the right to break-the-glass since the beginning. Although the break-the-glass will be recorded (a deterrent for abuses) Dr. Mario can still misuse that right. We need delegation for letting Dr. Mario gain the right to read dynamically and timely.

5 DELG

DELG in RBAC gives rise to extended models (e.g., see (Barka and Sandhu, 2000) (Barka and Sandhu, 2007) (Wang and Osborn, 2006)) that allow to transfer permissions or roles. Here, we mainly describe delegation of permissions: it permits a finer and more flexible control over the accesses to data, which makes access control more adaptable to the different situations characterizing healthcare.

DELG can be a *grant* (the delegator maintains the permission) or a *transfer* (the delegator loses the permission). They are permissions in themselves: so if $p = (op, ob)$ is a permission, $D^g_v(p) = (\texttt{grant}_v.op, ob)$ is the permission to grant to v permission p, while $D^t_v(p) = (\texttt{trans}_v.op, ob)$ is the permission to transfer to v permission p. A delegation can be revoked. $RK_v(p) = (\texttt{revoke}_v.op, ob)$ is the permission to revoke from v permission p.

Differently from BTG, nesting permissions, i.e., delegate a delegation, is meaningful here.

The RBAC's elements change accordingly. Precisely: $PRMS^D = \cup_{i=0}^{\infty} PRMS^D_i$, where the various $PRMS^D_i$ are defined as follows:

$$\begin{cases} PRMS^D_0 &= PRMS \\ PRMS^{D'}_{i+1} &= \{D^x_v(p) : p \in PRMS^{D'}_i, \\ & \quad v \in USERS,\ x \in \{\texttt{g}, \texttt{t}\}\} \\ PRMS^D_{i+1} &= PRMS^{D'}_{i+1} \cup \{RK_v(D^x_v(p)) : \\ & \quad p \in PRMS^{D'}_i, v \in USERS,\ x \in \{\texttt{g}, \texttt{t}\}\} \end{cases}$$

$PRMS^D$ is the language of all possible policies (in an instance of RBAC only specific policies will be considered) but it is not the richest possible: we have excluded from it policies expressing delegation of revoke. This is a simplification that we have made to keep our description clear to show how the management of delegation works, but still rich enough to compare its meaning with BTG. We consider to relax this constrain in future work.

PA^D is the extended relation PA that uses $PRMS^D$. The definition of policy set remains unchanged. We continue to call the policy set π, instead of π^D, to keep the notation lightweight. The authorization function $Auth^D_\pi : USERS \times PRMS^D \to \{\texttt{ff}, \texttt{tt}\}$ remains unchanged, but the policy set needs to be updated after any delegation/revocation.

In general, allowing delegation of roles (called "referral" in (Becker, 2005)) and a dynamically changing role hierarchy within a session may introduce inconsistencies if used uncontrollably. This calls for particular strategies of containment (e.g., see (Crampton and Khambhammettu, 2008)).

We can always refer to those strategies if we need them, but for the sake of keeping our delegation model simple we assume no dynamic change of roles. We are motivated not to present the richest model delegation management but rather to discuss delegation with respect to break-the-glass. However, even under this simplified assumption one needs to control what happens when permissions are dynamically assigned and then revoked. To trace this on the policy set, we use a *delegation history log*, a solution that is introduced and described in (Crampton and Khambhammettu, 2008). After any delegation, the delegation history log stores the delegator, the delegatee, and the permissions temporarily acquired (by the delegatee) or lost (by the delegator in case of a transfer). It changes again after a revocation. We also assume that the right to revoke is gained dynamically as a consequence of a delegation and kept until its use.

In the following we integrate directly the history log into π. As π changes, we called it *dynamic policy set* and we indicate it as π'. The dynamic policy set is in fact a multiset, at first initialized to have the same elements as π, then updated whenever a user delegates or revokes a permission. Updating π' is the side-effect of the `exec` procedure, which we assume to be the procedure called to execute the operation authorized by a policy. Here $\sqcup$ is the multiset union:

$\texttt{exec}(u, D^g_v(p))$**:** u grants permission p to v. Hence v gains p and u gains the permission to revoke that grant. $\pi' ::= \pi' \sqcup \{(v,p), (u, RK_v(p))\}$.

$\texttt{exec}(u, D^t_v(p))$**:** u transfers permission p to v. Hence v gains p but u looses it and gains the permission to revoke. $\pi' ::= \pi' \sqcup \{(v,p), (u, RK_v(p))\} \setminus \{(u,p)\}$.

$\texttt{exec}(u, RK_v(p))$**:** u revokes p from v. Hence v loses p and u recovers it but looses the permission to revoke: $\pi' ::= \pi' \sqcup \{(u,p)\} \setminus \{(v,p), (u, RK_v(p))\}$.

The authorization procedure $Auth^D_{\pi'}$ operates on the dynamic policy (multi) set π':

$$Auth^D_{\pi'} : USERS \times PRMS^D \to \{\texttt{ff}, \texttt{tt}\}$$

$$Auth^D_{\pi'}(u,p) \stackrel{\text{def}}{=} \begin{cases} \texttt{tt}, & \text{if } (u,p) \in \pi' \\ \texttt{ff}, & \text{otherwise} \end{cases} \quad (2)$$

Security Requirements. Doctors and patients should not loose control over who is accessing what data: we need to control that no one gains permissions that are not meant to have.

For example, we may want to avoid that someone could trigger a loop in the delegation chain as it happens when $Auth^D_\pi(u, D_u(D_u(p)))$ and $\neg Auth^D_\pi(u,p)$ or when $Auth^D_\pi(u, D_v(D_u(p)))$ and $\neg Auth^D_\pi(u,p)$. The first case suggests auto-assignment (we discuss thoroughly this case in §5.2; the second suggests u and v collude to let u obtain a permission. We qualify such circularities as *vicious*.

We can avoid vicious circularities by applying two different modalities of control: one is exerted, let say, *endogenously* to the policy set; the other one is exerted, let say, *exogenously* to it. In the endogenous control we limit the presence of polices in π that could lead to vicious circularities. We express the control as a security requirement, and a π compliant to it has no policies that could lead to potentially insecure situations. This way of controlling abuses may constrain the variety of situations we can model. In the exogenous control we do not limit policies in π. However, abuses are avoided by the enforcing mechanism that needs to look for environmental pieces of information to discern whether the request is an abuse or not before it can authorize the operation.

In this paper we opt for an endogenous control. We constrain the policies in π to avoid that one can delegate a permission s/he does not possess. Formally:

Requirement 1.

$$\forall (u,p) \in \pi,\ Auth^D_\pi(u, D_v(p)) \Rightarrow Auth^D_\pi(u,p)$$

Req 1 shapes how we can initialize the policy set π: it cannot be that u has permission $D_v(p)$ but not p. And, if p is a delegation (e.g., if p has the form $D_w(p')$), then u has to hold p' too, and so recursively, till the most innermost basic permission in p'.

The effect of Req 1 is that a permission p can be passed even through many hoops but there is always someone who holds p in π and stands at the root of a chain of delegation for that p. This is expressed by a Lemma (please recognize π' from π):

Lemma 1. *Let π satisfy Req 1. Then $\forall (u,p) \in \pi'$, $Auth^D_{\pi'}(u,p) \Rightarrow$ either $Auth^D_\pi(u,p)$ or there exists $v \neq u$ that initiates the chain of delegation that ends with u having p, and $Auth^D_\pi(v,p)$.*

Proof. The non-obvious case is when $Auth^D_{\pi'}(u,p)$ and $\neg Auth^D_\pi(u,p)$. Here, u must have received p from someone else, say w. Therefore there must have been an instance of π' (not necessarily the current one), such that $Auth^D_{\pi'}(w, D_u(p))$. If $Auth^D_\pi(w, D_u(p))$ then $Auth^D_\pi(w,p)$, for Req 1. Otherwise, w has received that right from someone else, say z. This backtracking eventually ends with a v that initiates the first delegation and that holds that right of delegation, in π. Then $Auth^D_\pi(v,p)$ must hold for Req 1. □

Req 1 justifies a review of the procedure that updates π' after a *transfer*: one should not be allowed to delegate a permission p that one has transferred, nor re-obtain it by creating vicious loops. This would violate the spirit of the transfer. Thus the effect of $\texttt{exec}(u, D^{\texttt{t}}_v(p))$ on the current π' must be changed in such a way to suspend u from starting a chain of delegation for p: this means removing all policies that are nested delegation of p that u may have in π'; in particular u looses $D^{\texttt{t}}_v(p)$. All such permissions will be restored after the transfer is revoked.

Req 1 may impose a restriction on what scenarios the model can handle. We cannot for example have policies that give to the manager of a hospital the right to delegate permission he does not have. Thus, if confronted with an emergency when a responsible doctor has left without having delegated a peer, the manager cannot delegate whichever doctor is available at that moment. However, relaxing Req 1 raises the need to have what we called an exogenous control of abuses. In our example, to understand whether the manager is legitimate in delegating cannot be decided only on the basis of his request; rather is the existence of a situation of emergency that justifies his acting. This condition can be checked in real-time before granting authorization, or *a posteriori* while auditing the manager's actions. There are a plethora of equally good solutions to implement this control. However, we have other reason to keep Req 1. It internalizes the security concerns in the policy set itself and makes it possible to compare the DELG and the BTG models without leaving any security implication out-of-bound. This is essential in §5.2 where we discuss the trust between delegator and delegatee only on the basis of the policies that bind them and without referring to their context. Moreover, in §6 we will see that the use of BTG will restore what seems lost by Req 1 in terms of expressibility, but without the need of any exogenous control.

5.1 Running Example (Continued)

The situation illustrated in §3.3 using DELG is represented in the following policy set (we write $\texttt{grant}/\texttt{transfer}(u, op(obj))$ instead of $(\texttt{grant}_u/\texttt{transfer}_u.op, obj)$):

$$\pi = \{(\texttt{DrJohn}, (\texttt{read}(\texttt{blood\_test})));$$
$$(\texttt{DrJohn}, (\texttt{grant}(\texttt{Michel}$$
$$(\texttt{transfer}(\texttt{DrMario}, (\texttt{read}(\texttt{blood\_test}))))))))\}$$

First, let us stress that the decentralized paradigms we advocate for healthcare suggests that is Dr. John who initializes π. We imagine him setting the policies about the data he is managing. He is the one who should have the right to initiate a chain of delegation that sees Michel to transfer to Dr. Mario the right to read. This use of delegation suggests that is discretionary to Michel to decide *when* to transfer. She may or may not follow Dr. John's instructions on the matter. However, there is another way to define the same situation by using auto-delegation.

$$\pi = \{(\texttt{DrJohn}, (\texttt{read}(\texttt{blood\_test})));$$
$$(\texttt{DrJohn}, (\texttt{grant}(\texttt{Michel},$$
$$(\texttt{transfer}(\texttt{DrMario},$$
$$(\texttt{grant}(\texttt{DrMario}, (\texttt{read}(\texttt{blood\_test}))))))))))\}$$

This version is slightly different. Michel can transfer Dr. Mario the right to auto-delegate reading. How this setting differs from what is presented in §4.1, where Dr. Mario was given the right to break-the-glass and read, is discussed in the next section. Note that assigning the right to read to Dr. Mario would imply letting him have full read-control over Rachel's data despite her not being Dr. Mario's patient.

5.2 Auto-delegation and BTG

For a user u, auto-delegation is a permission of the form $(u, D^g_u(p))$ that u is authorized to execute. Auto-delegation can only be used as auto-grant, since auto-transfer introduces the contradictory situation where u gains and looses p at the same time.

Auto-delegation is not ruled out by Req 1, but it constrains how u can get that permission. Namely, u can possess auto-delegation of p because of (a) u has already that right in π; (b) u has been delegated to have it by delegation; however, in the chain of delegation there is someone who, eventually, stholds p.

In case (a), Req 1 says that u must also hold p in π, removing the need of auto-delegation.

In case (b), Lemma 1 says that the root of the delegation chain that ends in u must hold p in π.

That pointed out, we would like to understand whether auto-delegation differs from BTG and, if it does, what elements make them different. At a first sight, some difference exists: having BTG in the initial policy set π is meaningful, whereas Req 1 suggests this being superfluous for auto-delegation (case (b) above). Deciding BTG requires an interaction with the user, whereas auto-delegation does not. Except these minor differences the two concepts look the same.

Another way to shed light on the matter is by asking what situations we can model with one that we cannot with the other. Crampton and Morriset (Crampton and Morisset, 2011) have shown that auto-delegation can simulate break-the-glass. Auto-delegation on p is authorized only if no one else with at least the same role as the requester is logged in the system. This use of auto-delegation supersedes BTG.

However, BTG has been explicitly introduced to handle "exceptional situations". BTG is meant to be used parsimoniously and signals, in itself (i.e., endogenously), an exception. Implementing BTG as auto-delegation restrains auto-delegation for this exceptional usage only, even though auto-delegation does not necessarily flag for any exceptional situations having occurred and although auto-delegation could be used to model also different situations. In fact, despite having the same consequences as auto-delegation (i.e., that one circularly obtains a non-possessed permission), BTG should not be used routinely. So, one way to inquire whether BTG is superfluous is by understanding whether auto-delegation must be necessarily interpreted as an exceptional violation —as BTG is– or whether there are other uses of auto-delegation than that which matches BTG. If auto-delegation were necessarily interpreted as an exceptional violation, then auto-delegation would coincide with BTG. Delegation would became the only policy really needed and there would be nothing more to discuss. But, if auto-delegation is not necessarily interpreted as an exceptional violation, then we can consider to keep BTG to handle violations, and free auto-delegation for the other uses that this policy may have.

We have an argument in favour of this latter situation. Req 1 suggests that auto-delegation can be a transferable right (case (a) above) whereas nothing indicates that it must be tolerated only under "exceptional situations". Granting auto-delegation may be done for other reasons. One of such reason is when there is enough trust between delegator and delegatee to induce the delegator to let the delegatee auto-delegate a right. For instance, only a doctor that trusts her assistant delegates her the permission to auto-grant reading a document. In fact, at least the RBAC models described so far, when the assistant gains the permission to read by auto-assignment, the doctor has no power to revoke it. Only she can decide to auto-revoke it. This use of auto-delegation suggests a degree of trust that is higher than that occurring when

the doctor delegates directly the right to read, because in this case the doctor retains the right to revoke the delegated permission. At the same time, both situations suggest more trust then that in place when the doctor grants a BTG on reading. Breaking-the-glass is still a violation but it is tolerated, expected to be infrequent, and occurring under well justified circumstances.

According to this interpretation, if we use auto-delegation to implement BTG we miss the possibility to model the various situations just exemplified. But if we let DELG co-exist with BTG then we are left with two modalities to delegate auto-assigning a right: one via BTG and the other via auto-delegation. Even though any decision should be audited and the reason of it questioned, in the first case the invocation of BTG is justified in the presence of exceptional or urgent situations while the second does not require that motivation; despite it can be question, it rests at the delegatee's discretion and responsibility. The resulting RBAC looks richer in possibilities.

6 AN RBAC WITH BTG AND DELG

Thus, we assume that BTG and DELG can co-exist as different policies and we study a version of RBAC that includes both. In so doing, we intend to verify whether we obtain a different model than the one that uses only delegation and where break-the-glass can be simulated as auto-delegation. Our criterion of "being different" is defined in terms of having different security requirements.

The language of this new RBAC's policy is $PRMS^{BD} = \cup_{i=0}^{\infty} PRMS_i^{BD}$, where the $PRMS_i^{BD}$ are iteratively defined as in Table 2. In $PRMS^{BD}$ there are policies we consider useless: those expressing chains of *BTG*, as we discussed in § 4; those expressing loops of auto-delegation, as discussed in § 5.2. Besides, we consider useless those made of interleaved BTG and DELG that express loops of auto-assignment e.g., $(D_u^g(BTG(D_u^g(p))))$ used for u.

The RBAC relations change accordingly with the new permission set. PA^{BD} is the relation PA that uses $PRMS^{BD}$. The new permissions can be subjected to new obligations (but, as we have done so far, we do not include them explicitly in the model).

The definitions of π and π', which now operate on the extended relations, remain the same, but updating π' needs some discussion. In section §5 we pointed out that, when u runs $\texttt{exec}(u, D_v^{\mathtt{t}}(p))$, i.e., a transfer, u looses temporarily p and the right to delegate p. This is still valid here at least for what concerns nested delegations. Besides, u looses (possibly nested) delegation of p that are conditioned by *BTG*: since u does not possess p any more, it cannot delegate p even by breaking-the-glass on delegation. The only exception we may consider (but we have not a strong argument for it) are delegations that would be allowed, if executed, to let u re-obtain p. This seems to be in the spirit of BTG, i.e., a tolerated but controlled violation whose effect is letting someone re-obtain p. It seems an alternative way to revoke that right to a delegatee, but happening under the conditions dictated by BTG. If this argument holds there is no reason to suspend u from policies like e.g., $BTG(D_v^g(D_u^g(p)))$ or $(D_v^g(BTG(D_u^g(p))))$ after u transfers p. The choice is left to the discretion of the policy maker.

The definition of $Auth_{\pi}^{BD}$ and of $Auth_{\pi'}^{BD}$ remains unchanged but use the new π and π'. Formally:

$$Auth_{\pi'}^{BD} : USERS \times PRMS^{BD} \rightarrow \{\texttt{ff}, \texttt{tt}\}$$

$$Auth_{\pi'}^{BD}(u,p) \stackrel{\text{def}}{=} \begin{cases} \texttt{tt}, & \text{if } (u,p) \in \pi' \\ \texttt{ff}, & \text{otherwise} \end{cases}$$

$Auth_{\pi'}^{BD\prime}$ is the extended function requiring the answer of the user on the break the glass policy:

$$Auth_{\pi'}^{BD\prime} : USERS \times PRMS_{\infty'}^{BD} \times \{\mathrm{y}, \mathrm{n}\} \rightarrow \{\texttt{ff}, \texttt{tt}\}$$

$$Auth_{\pi'}^{BD\prime}(u,p,a) \stackrel{\text{def}}{=} \begin{cases} \texttt{tt}, \text{if } Auth_{\pi'}^{BD}(u,p); \\ \texttt{ff}, \text{if } \neg Auth_{\pi'}^{BD}(u,p) \wedge (a = \mathrm{n}); \\ Auth_{\pi'}^{BD}(u, (BTG(p))), \\ \quad \text{if } \neg Auth_{\pi'}^{BD}(u,p) \wedge (a = \mathrm{y}); \end{cases}$$

$PRMS^{BD}$ is the set of all policies that do not start with a BTG, that is $PRMS^{BD} = PRMS_{\infty'}^{BD} \setminus \cup_{i=0}^{\infty} PRMS_{i''}^{BD}$.

Security Requirements. Since the model includes DELG, Req 1 holds in this extension too. It remains to state the requirements that emerge because of BTG and DELG's interleaving. The criterion we used to identify the requirement is the same: to initialize π to avoid that permissions appear from nowhere.

Requirement 2.

$$\forall (u,p) \in \pi, \\ Auth_{\pi}^{BD}(u, BTG(D_v(p))) \Rightarrow Auth_{\pi}^{BD}(u,p)$$

Req 2 says that, in π, when u is assigned the right to break-the-glass and delegate a permission then u must hold that permission. This is because BTG is on delegation and when actually u violates the policy and delegates p, u is the root of the chain and must hold that permission. Req 2 avoids the creation of permissions and preserves the BTG's goal: obtaining

Table 2: The policy language combining BTG and DELG.

$$\begin{cases} PRMS^{BD}_{0'} = PRMS^{D} \\ PRMS^{BD}_{0''} = PRMS^{B} \\ PRMS^{BD}_{0} = PRMS^{BD}_{0'} \cup PRMS^{BD}_{0''} \end{cases} \qquad \begin{cases} PRMS^{BD}_{(i+1)'} = \{D(p) : p \in PRMS^{BD}_{i}\} \\ PRMS^{BD}_{(i+1)''} = \{BTG(p) : p \in PRMS^{BD}_{i'}\} \\ PRMS^{BD}_{i+1} = PRMS^{BD}_{(i+1)'} \cup PRMS^{BD}_{(i+1)''} \end{cases}$$

p must be occurring under a controlled violation of the policy enforcement and not by vicious transfers that short-cut BTG. This would happen, for example if u where entitled, in π, to $BTG(D_v(D_u(p)))$ without holding p already. We stress that we are talking about the initial policy set π; in the dynamic one, π', such situations can happen via a chain of delegations when at the root there is someone who holds p.

This is indeed an interesting security property that comes from Req 2 and Requirement 1. It is an extension of Lemma 1.

Lemma 2. *Let π satisfy Reqs 1–2. $\forall(u,p) \in \pi'$ such that $Auth^{BD}_{\pi'}(u,p) \wedge \neg Auth^{BD}_{\pi}(u,p)$ then $\exists w$ that initiates the chain of delegation that ends with u having p, and $Auth^{BD}_{\pi}(w,p)$.*

Proof. (Concise). If u is not authorized to p from some policy in π, it means that u has been delegated p by someone. This someone either has the authorization to delegate p or the possibility to break the glass and delegate p. If he had those rights because of some policy in π, then the thesis is proved for Req 2. If not, he himself has been delegated. This creates a chain that is finite, since users are finite in number and policies cannot be infinitely long. Let w be the one who first starts to delegate. This can be because he has the right to delegate or BTG on delegation; then, for Req 2 or Req 1 w must have p. □

We observe that, after integrating BTG with DELG, our security requirement is no more restrictive in terms of expressiveness. The situation in § 5 seeing an hospital manager delegating to whichever doctor was available in case of an emergency (which was hard to model using delegation in a policy set compliant with Req 1) can be now easily modelled by letting the manager having BTG right on delegation.

In this extended model there is another desirable property:

Property 1.

$$\forall(u,p) \in \pi,\ Auth^{BD}_{\pi}(u,D_v(p)), v \neq u \Rightarrow Auth^{BD}_{\pi}(u,D_v(BTG(p)))$$

Property 1 suggests that whoever has permission to delegate p can delegate less permissive permissions than p, i.e., BTG on p and auto-delegation on p. However, this property has no consequences on the security of the model as Reqs 1 and 2 have. We do not impose it as mandatory, but we leave to the discretion of the policy maker to following it in order to lead to more flexible decisions. In healthcare this is often desirable for decisions are often taken on the basis of need-to-know accesses.

6.1 Running Example (Epilogue)

We can now give a more satisfactory implementation of policies for the scenario described in §3.3. Dr. John delegates his assistant to transfer the right to read to Dr. Mario. However, since Dr. John does not want Michel to abuse this right but only use it as an exceptional measure, he imposes a BTG of that right instead of a full permission. The risk of abuses should be more deterred than in the previous example using delegation only. As soon as Dr. John is back, he can revoke the assigned permissions. Note that Dr. John could have decided to let Michel transfer to Dr. Mario the permission to auto-delegate the reading instead of the permission to read directly. This requires an higher level of trust between the doctors.

The policies are set around Dr. John. He can start the chain of delegation before he leaves. Below, we write more compactly $op(obj)$ instead of (op,obj), and $\texttt{grant}(u,op(obj))$ instead of $(\texttt{grant}_u.op,obj))$:

$$\begin{aligned}\pi = \{&(\texttt{DrJohn},(\texttt{read}(\texttt{blood\_test}))); \\ &(\texttt{DrJohn},(\texttt{grant}(\texttt{Michel},(\texttt{btg}(\texttt{transfer} \\ &\quad(\texttt{DrMario},(\texttt{read}(\texttt{blood\_test}))))))))\}\end{aligned}$$

Such set, however, violates Req 1. In order to grant that right to his assistant, Dr. John must have the permission to BTG and transfer his colleague as well. This triggers also Req 2 suggesting that Dr. John must have the right to read himself, which he already has. A compliant policy set is therefore the following:

$$\begin{aligned}\pi = \{&(\texttt{DrJohn},(\texttt{read}(\texttt{blood\_test}))); \\ &(\texttt{DrJohn},(\texttt{grant}(\texttt{Michel},(\texttt{btg}(\texttt{transfer} \\ &\quad(\texttt{DrMario},(\texttt{read}(\texttt{blood\_test})))))))); \\ &\quad(\texttt{DrJohn},(\texttt{btg}(\texttt{transfer} \\ &\quad\quad(\texttt{DrMario},(\texttt{read}(\texttt{blood\_test}))))))\}\end{aligned}$$

Then it comes the moment when Dr. John leaves. He grants his assistant with the necessary right to act on his behalf. This will change dynamically the policy set as follows:

$$\pi' = \pi \cup \{(\texttt{DrJohn}, \texttt{revoke} \\ (\texttt{Michel}, (\texttt{btg}(\texttt{transfer}, \\ (\texttt{DrMario}, (\texttt{read}, \texttt{blood\_test}))))); \\ (\texttt{Michel}, (\texttt{btg}(\texttt{transfer} \\ (\texttt{DrMario}, (\texttt{read}, \texttt{blood\_test})))))\}$$

Michel does not necessarily have the right to read, i.e., Req 2 does not apply here. But this is fine because Michel has received the right to break-the-glass and transfer from Dr. John who is at the root of the delegation chain. Then it comes the time when Dr. Mario needs accessing Rachel's health records. Michel can break the glass and transfer to him the right to read Rachel's blood test. When Michel breaks the glass and transfers the policy set becomes:

$$\pi' = \pi' \cup \{(\texttt{DrMario}, (\texttt{read}, \texttt{blood\_test})); \\ (\texttt{Michel}, (\texttt{revoke}(\texttt{DrMario}, (\texttt{read}, \texttt{blood\_test}))))\}$$

Dr. Mario has now the right to read until Michel revokes it.

7 CONCLUSIONS AND FUTURE WORK

In healthcare, accessing medical data is particularly critical because most of the records contain sensitive information. This justifies a strict control of accesses to the data. However, the enforcement should be flexible to work in circumstances of emergency, which stretch the applications of the access control rules.

Features such as *break-the-glass* and *delegation* have been introduced rightly to cope with such situations. They work best in a decentralized architecture without a central authority that distributes accesses and takes decisions, because these tasks are usually left to healthcare practitioners who face often unpredictable situations.

So far, no RBAC model combines *break-the-glass* and *delegation*. It is even unclear whether these two permissions are independent. Auto-delegation has been shown to approximate a break-the-glass behaviour, but the question whether the two permissions are really both necessary or not has not been stated not answered. We have deepened the subject and given an answer to it. We showed that there is an interpretation of auto-delegation which can coexist with break-the-glass without overriding it but rather making the resulting access control model richer. In the domain of healthcare access control this coexistence is justified and we provided logical arguments in its support.

As a result we proposed an access control model of the RBAC family that integrates break-the-glass and delegation. We defined the authorization function and the tools to keep the policy set updated after a delegation (a grant or a transfer) or/and a break-the-glass action. We studied what security requirements a policy set that handles both permissions must have. These requirements can be used with a policy checker to help policy makers —doctors and patients— write secure-by-design policies.

The functionalities introduced in this paper (i.e., the authorization function and the security requirements) are meant to be implemented and integrated into an access control system. The high-level architecture we envision is depicted in Fig. 1. The authorization functions can be integrated into the enforcement point (Authorization/Enforcement), together with the procedures that keep π updated as we explained in §6. Our requirements can be instead implemented as a front-end (Policy Checker) that helps a policy maker —a patient or a doctor— to write security policies on the data s/he owns or is a guardian for. This front-end can be employed in two complementary ways: (1) as a policy checker that evaluates whether an already defined policy set contains potential vicious circularities; (2) as a policy helper that suggests in real time the missing policies that make the policy set compliant with the requirements. In this way, the policy maker can concentrate on designing the core permissions, according to the access control high-level strategy s/he is implementing, while having the subsidiary security-compliant policies set automatically.

As proof-of-concept we have prototyped the policy checker and the authorization function in Ocaml (http://ocaml.org) and used it to validate our requirements against samples of policy sets[1]. One of them has been reported here as a running example.

This work has limitations that we intend to address in the future. First, we simplified considerably the delegation model by assuming no delegation of roles and no delegation of revocation. In the scope of this paper's goal this simplification was justified to keep focus on comparing delegation and break-the-glass as policies. Considering a fully fledged management of delegation would have complicated the section on delegation with no benefit for the part where we compare the two types of policies. However, after we proved that delegation and break-the-glass can coexist, considering a complete treatment of delegation becomes justifiable. Adding delegation of roles it is mainly a technical problem, since we can adopt standard solutions from the literature (e.g., (Zhang et al., 2003)). Adding delegation of revocation should not require much effort, except restoring a fully informative del-

[1]The Ocaml code is available on request.

Figure 1: Our access control system and our requirements in support to a policy maker.

egation history log (separated and not integrated in π as we did here) to be able to return a permission p to the original owner/delegator when it is revoked from the delegatee by a third-party.

ACKNOWLEDGEMENT

We are grateful to several anonymous reviewers. In different rounds of evaluation, their positive critics helped us to improve our arguments and the relevance of our contribution.

REFERENCES

Barka, E. and Sandhu, R. (2000). Framework for role-based delegation models. In *Proc. of 6th Ann. Conf. on Computer Security Applications (ACSAC'00)*, pages 168–176.

Barka, E. and Sandhu, R. (2007). Framework for Agent-based Role Delegation. In *Proc. of the IEEE Int. Conf. on Communications (ICC'07*, pages 1361–1367

Becker, M. Y. (2005). A Formal Security Policy for an NHS Electronic Health Record Service. Technical Report UCAM-CL-TR-628, University of Cambridge.

Brucker, A. D. and Petritsch, H. (2009). Extending Access Control Models with Break-glass. In *Proc. of the 14th ACM Symposium on Access Control Models and Technologies (SACMNAT '09)*, pages 197–206. ACM.

Crampton, J. and Khambhammettu, H. (2008). Delegation in role-based access control. *Int. J. of Information Security*, 7(1):123–136.

Crampton, J. and Morisset, C. (2011). An auto-delegation mechanism for access control systems. In *Proc. of Security and Trust Management (STM)*, volume 6710 of *LNCS*, pages 1–16. Springer Berlin Heidelberg.

Ferreira, A., Chadwick, D., Farinha, P., Correia, R., Zao, G., Chilro, R., and Antunes, L. (2009). How to Securely Break into RBAC: The BTG-RBAC Model. In *Proc. of 5th Ann. Conf. on Computer Security Applications Conference (ACSAC'09)*, pages 23–31.

Ferreira, A., Cruz-Correia, R., Antunes, L., and Chadwick, D. (2007). Access control: how can it improve patients'healthcare? In *Medical and Care Compunetics*, volume 127 of *Studies in Health Technology and Informatics*, pages 65–76.

Ferreira, A., Cruz-Correia, R., Antunes, L., Farinha, P., Oliveira-Palhares, E., Chadwick, D., and Costa-Pereira, A. (2006). How to break access control in a controlled manner. In *Proc. of 19th IEEE Int. Symp. on Computer-Based Medical Systems (CBMS)*, pages 847–854.

Hasebe, K. and Mabuchi, M. (2010). Capability-role-based delegation in workflow systems. In *Proc. of IEEE/IFIP 8th Int. Conf. on Embedded and Ubiquitous Computing (EUC 10)*, pages 711–717.

ISO/TS (2009). ISO/TS 13606-4: Health informatics - electronic health record communication - part 4: Security.

Krautsevich, L., Martinelli, F., Morisset, C., and Yautsiukhin, A. (2012). Risk-Based Auto-delegation for Probabilistic Availability. In *Data Privacy Management and Autonomous Spontaneus Security*, volume 7122 of *LNCS*, pages 206–220. Springer Berlin Heidelberg.

Li, M. and Wang, H. (2008). ABDM: An extended flexible delegation model in RBAC. In *Proc. of the 8th Int. Conf. on Computer and Information Technology (CIT 2008)*, pages 390–395.

Maw, H., Xiao, H., Christianson, B., and Malcolm, J. (2014). An evaluation of break-the-glass access control model for medical data in wireless sensor networks. In *Proc. of IEEE 16th Int. Conf. on e-Health Networking, Applications and Services (Healthcom)*, pages 130–135.

Rajesh, K. and Nayak, A. (2012). Modified BTG-RBAC model for SaaS. In *Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference on*, pages 77–81.

Rissanen, E., Firozabadi, B., and Sergot, M. (2006). Towards a Mechanism for Discretionary Overriding of Access Control. In *Security Protocols*, volume 3957 of *LNCS*, pages 312–319. Springer Berlin Heidelberg.

Rostad, L. and Edsberg, O. (2006). A study of access control requirements for healthcare systems based on audit trails from access logs. In *Proc. of the 22nd Annual Computer Security Applications Conference (ACSAC '06)*, pages 175–186.

Sandhu, R., Coyne, E., Feinstein, H., and Youman, C. (1996). Role-based access control models. *Computer*, 29(2):38–47.

Wainer, J. (2005). A fine-grained, controllable, user-to-user delegation method in RBAC. In *Proc. of 10th ACM Symp. on Access Control Models and Technologies (SACMAT '05)*, pages 59–66. ACM Press.

Wang, H. and Osborn, S. L. (2006). Delegation in the role graph model. In *Proc. of 11st ACM Symp. on Access Control Models and Technologies (SACMAT '06)*, pages 91–100. ACM.

Zhang, X., Oh, S., and Sandhu, R. (2003). PBDM: A Flexible Delegation Model in RBAC. In *Proc. of the 8th ACM Symp. on Access Control Models and Technologies (SACMAT '03)*, pages 149–157, New York, NY, USA. ACM.

Zhao, G., Chadwick, D., and Otenko, S. (2007). Obligations for Role Based Access Control. In *Proc. of the 21st Int. Conf. on Advanced Information Networking and Applications Workshop (AINAW'07)*, volume 1, pages 424–431.