A first step towards formal verification of security policy properties for RBAC

Michael Drouineaud
Email: mdruid@tzi.de

Maksym Bortin (Email: maxim@tzi.de)
Paolo Torrini (Email: paolot@tzi.de)
Karsten Sohr (Email: sohr@tzi.de)
Bremen Institute of Safe and Secure Systems
Department of Mathematics and Computer Science
Universität Bremen
Bibliothekstr. 1
D-28359 Bremen
Germany

Abstract

Considering the current expansion of IT-infrastructure the security of the data inside this infrastructure becomes increasingly important. Therefore assuring certain security properties of IT-systems by formal methods is desirable. So far in security formal methods have mostly been used to prove properties of security protocols. However, access control is an indispensable part of security inside a given IT-system, which has not yet been sufficiently examined using formal methods. The paper presents an example of a RBAC security policy having the dual control property. This is proved in a first-order linear temporal logic (LTL) that has been embedded in the theorem prover Isabelle/HOL by the authors. Thus the correctness of the proof is assured by Isabelle/HOL. The authors consider first-order LTL a good formalism for expressing RBAC authorisation constraints and deriving properties from given RBAC security policies. Furthermore it might also be applied to safety-related issues in similar manner.

1. Introduction

1.1. General thoughts

Since IT-technology is pervading our daily life more and more, data security has become a topic of increasing relevance in our society. Unfortunately there is no general definition of security for all scenarios. Consider e.g. an electronic transaction between a customer and a merchant. The customer then may expect, that a secure transaction grants him anonymity, while the merchant wishes, that it should always be possible to identify any customer who ordered some product in a way a court will accept as proof. A police officer on the other hand will regard an electronic transaction as secure if it allows him to trace electronic money. Obviously it will be hard for these three parties to find a common definition of security for electronic transactions. Even if all the involved partners agree on the desired properties of security as e.g. email-provider and client, who both wish to protect their communication against eavesdropping or other manipulations by third parties, there may still remain some details as for example the presumptions about the behaviour of the clients to be settled. If one assumes, that any client will always close his browser after leaving his email account, a logout procedure may not be necessary. But if this precondition does not hold the provider should make sure, that it is not possible to reactivate an email account which has been left by its user via the backtracking function of the browser without entering the password again. Otherwise the service of the email-provider may no longer be regarded as secure.

This illustrates, that security has first to be carefully defined before one can reason about it. Thus formal methods can be a way to achieve an exact (and consistent) definition of security for a given scenario. In fact various properties of some security protocols (SET, TLS, etc.) (cf [15, 5]) have been proved using formal methods. This included of course an exact specification of these protocols and resulted in precisely specified proven properties. Well-known formalisms for the examination of security protocols are e.g. the Spi Calculus or the BAN logic (see [3, 7]).

The topic of this paper is the analysis of security policies for role-based access control (RBAC) by formal methods. A *security policy* for access control is a set of rules determining the desired behaviour of the IT-system with re-

spect to granting user requests for the application of access operations (e.g. read, write, append, etc.) to relevant data. RBAC is a particularly well-suited method of access control (cf [9]) for hierarchical organisations such as banks, authorities, hospitals, etc. It is also apt for the use in distributed systems. The following subsection illustrates, why this topic in our opinion is an important part of security.
Earlier approaches to the examination of access control using formal methods (e.g. the ABLP logic) can be found in [2].

1.2. Practical relevance

One of the results of our inquiries was, that a hospital in Bremen using SAP software with RBAC capability needed already more than 25 different medical roles. It is rather obvious, that the task of defining a sound security policy for such IT-systems is most likely beyond the scope of the human mind just because of the amount of roles alone. Thus automated support (based on formal methods) for this kind of problem should be very welcome.

1.3. Structure of the paper

The paper consists of five sections (including the introduction). In the following section we will explain RBAC and give some information about the mentioned LTL and Isabelle/HOL. You will also find some short remarks about the embedding of the LTL in Isabelle/HOL. At the end of this section our encoding (cf RBAC.thy at [1]) of RBAC (using the LTL theory) in Isabelle/HOL is described in detail.

The third section then gives a rather detailed presentation of an RBAC security policy encoded in Isabelle/HOL that has the dual control property. The authors have managed to find a proof for this *which has been verified by Isabelle/HOL*. The section closes with a short summary of this proof, since the proof itself is easily understandable and straightforward unlike the corresponding Isabelle file Example.thy (see [1]).
In the fourth section the advantages of theorem provers will be illustrated by a second example (Example2.thy at [1]).The fifth section will present our conclusions and an outlook. The files containing our proofs verified by Isabelle/HOL will soon be made available for download at [1].

2. RBAC and LTL

Role-based access control (RBAC) has received considerable attention as a promising alternative to traditional discretionary and mandatory access control. One reason for this increasing interest was an extensive field study [9] carried out by the National Institute of Standards and Technology (NIST) which pointed out that in practice permissions are assigned to users according to their roles/functions in the organization (governmental or commercial). In addition, the explicit representation of roles greatly simplifies the security management and makes possible to use well-known and time-honored security principles like separation of duty and least privilege [16]. Furthermore an RBAC standard [10] has been proposed, which is based on the RBAC96 model introduced by Sandhu et al. [16].
The RBAC96 model (adapted for our scenario) has the following components (see figure 1):

Users – set of users, Roles – set of roles, P – set of permissions
$UA \subseteq Users \times Roles$ (user assignment)
$RH \subseteq Roles \times Roles$ is a partial order also called the role hierarchy or role dominance relation written as $\leq$. We will not use role hierarchies in our examples.
$PA \subseteq Roles \times P$ (permission assignment)

P is the set of ordered pairs of operations and objects. In the context of security and access control all resources accessible in an IT-system (e.g. files , database tables, etc.) are named by the notion *object*. An *operation* is an active process applicable to objects (e.g. read, write, append, etc.). The relation PA assigns to each role a subset of P. So PA determines for each role the operation(s) it may execute and the object(s) to which the operation in question is applicable for the given role. Thus any user having assumed this role can apply an operation to an object if the corresponding ordered pair is an element of the subset assigned by PA to the role.

Furthermore we omit the session concept, which is a part of the RBAC96 model, for reasons of simplicity.

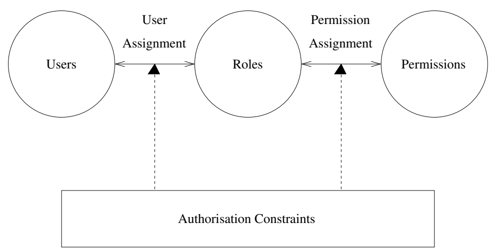

Figure 1. The RBAC model

The *authorisation constraints* depicted in figure 1 are additional conditions imposed on RBAC (e.g. restrictions on the relations UA or PA, temporal constraints, mutually exclusive roles, etc.).

2.1. LTL for RBAC

To express these authorisation constraints we define a single-sorted (in contrast to [12] first-order LTL with infinite future and finite past. The axiomatisation of this LTL is based on the one given by Goldblatt in [11] for propositional LTL. Our first-order LTL is an extension of this preserving the usual semantics.
Function and predicate symbols are partitioned into *rigid* and *flexible* symbols: the former do not change over time, while the latter may vary. Sentences are the usual first-order sentences built from equations, predicate applications and logical connectives and quantifiers $\forall$, $\exists$. Additionally, we have the modalities BOX (always in the future), DMD (sometimes in the future), NEXTA (in the next world, if a such exists) and NEXTE (in the next world). The corresponding past modalities are PBOX, PDMD, PNEXTA and PNEXTE. From the literature the reader may be familiar with the symbols $\Box$ (for BOX), $\Diamond$ (for DMD) and so on. Our choice is due to the intention to have a convenient encoding of the first-order LTL for the theorem prover Isabelle/HOL.The encoding is based on previous work by Torrini [18]. Thus first-order LTL is particularly well-suited to express temporal dependencies of e.g. workflows explicitly. Thus it may be regarded as an improvement of the formalism given in [6]. An extensive documentation of Isabelle/HOL [14] is available via the Internet. Having done this we can encode RBAC (without sessions and role hierarchies) in Isabelle/HOL as follows:

theory `RBAC = LTL :`

2.1.1. Predicates for sorts, operations and relations between users, roles and operations

consts

```
  "Role"      :: "'a => 'a wff"
  "User"      :: "'a => 'a wff"
  "Object"    :: "'a => 'a wff"
  "Operation" :: "'a => 'a wff"

  "DOMAIN"    :: "'a => 'a => 'a wff"
  "RANGE"     :: "'a => 'a => 'a wff"
  "Eval"      :: "'a => 'a => 'a => 'a wff"

  "PA"         :: "'a => 'a => 'a => 'a
wff"
  "UA"         :: "'a => 'a => 'a wff"
  "AUTH"       :: "'a => 'a => 'a => 'a
wff"
  "EXEC"       :: "'a => 'a => 'a => 'a
wff"
  "ACTIVE_FOR" :: "'a => 'a => 'a wff"
```

axioms

```
 DOM_ax : "H |- AL1 x y. (DOMAIN x y ->
(Operation x && Object y)) "
 RAN_ax : "H |- AL1 x y. (RANGE  x y ->
(Operation x && Object y)) "
 EVAL_ax: "H |- AL1 x y z. (Eval x y z ->
(Operation x && Object y && Object z &&
DOMAIN x y && RANGE x z))"
 PA_ax  : "H |- AL1 x y z. (PA x y z ->
(Role x && Operation y && Object z &&
DOMAIN x z)) "
 UA_ax  : "H |- AL1 x y. (UA x y -> (User x
&& Role y)) "
 AUTH_ax  : "H |- AL1 x y z. (AUTH x y z
-> (User x && Operation y && Object z &&
DOMAIN y z))"
 EXEC_ax  : "H |- AL1 x y z. (EXEC x y z
-> (User x && Operation y && Object z &&
DOMAIN y z)) "
 ACTIVE_FOR_ax  : "H |- AL1 x y. (ACTIVE_FOR
x y -> (User x && Role y)) "
```

As already mentioned our LTL is a single-sorted logic (sort `'a`). Predicates are then defined as functions mapping arguments of type `'a` to well-formed formulae of type `'a wff`. In contrast to many functional programming languages Isabelle insists on explicit declaration of all functions (keyword **consts**). The unary predicate *User* (see above) is then defined as unary function from `'a` to `'a wff` and so on. Now we define the predicate *User* applied to a term *t* to be true if and only if *t* is a user (we assume *t* to be a constant). Thus the "sorts" users, roles, operations and objects are defined by unary predicates of the same name. *DOMAIN* and *RANGE* are binary predicates, *Eval* is a predicate with three arguments. If now the predicate *DOMAIN* is true for a pair (x,y), then it is ensured by the axiom *DOM_ax*, that x is an operation and y is an object, because the "sorts" operations and objects are defined by the unary predicates *Operation* resp. *Object* (the arrow symbol "$->$" denotes logical implication). The same holds also for the predicate *RANGE* (cf. *RAN_ax*). According to axiom *EVAL_ax* the predicate *Eval* being true for a triple (x,y,z) implies the following:

- x is an operation
- y is an object
- z is an object
- the predicate *DOMAIN* is true for the pair (x,y)
- the predicate *RANGE* is true for the pair (x,z)

As the reader may already have suspected, the purpose of the predicates *DOMAIN*, *RANGE*, *Eval* is to describe the behaviour of operations. If there is e.g. an operation "append" whose domain are pairs of writable files and strings (f,s), then the predicate *DOMAIN* will be true for the pair (append,(f,s)) if and only if the pair (f,s) is in the domain of the operation "append". The range of this operation are then writable files g, i.e. the predicate *RANGE* is true if and only if the file g is in the range of the operation "append". Finally the predicate *Eval* shall be true for a triple (append,(f,s),g) if and only if g is the result of applying the operation "append" to the pair (f,s), i.e. the file obtained by appending the string s to the file f.

The predicates *PA* and *UA* define relations between roles, operations and objects resp. between users and roles. PA is true for a triple (r,op,obj) if and only if the role *r* is allowed to apply the operation *op* to the object *obj*. UA assumes the value true for a pair (u,r) if and only if the user *u* is assigned to the role *r*. The predicate *ACTIVE_FOR* is true for such a pair if and only if the user *u* has activated the role *r*.

The truth values of the predicate *AUTH* indicate if for a given triple (u,op,obj) the user *u* is allowed to apply the operation *op* to the object *obj*. If AUTH is true for a triple (u,op,obj), then the truth value of the predicate *EXEC* tells if the user exercises this permission .

For the sake of simplicity we demand, that UA and PA are **rigid** predicates. In general the truth values of predicates need not be fixed, i.e. a predicate may have different truth values at various points of time, even if it is applied to the same argument(s). But rigid predicates will have the same truth values at all points of time as long as they are applied to the same argument(s). Consequently a conjunction of rigid predicates being true at some point of time will hold at any point of time (for the same arguments). Thus a user u who is assigned to a role r at some point of time is assigned to this role at all point of times. In the following axioms the symbols *UP*, *RP*, *OpP* and *ObP* denote arbitrary rigid unary predicates applicable to users, roles, operations and objects.

2.1.2. Axioms for rigid predicates

axioms

```
rigid_PA : " H |- ((PA r operat obj) &&
((RP::'a => 'a wff) r) && ((OpP::'a => 'a
wff) operat) &&((ObP::'a => 'a wff) obj))
              -> ((PBOX ( (PA r operat obj)
&& (RP r) && (OpP operat) && (ObP obj) ))
               && (BOX  ( (PA r operat obj)
&& (RP r) && (OpP operat) && (ObP obj) )) )
"

rigid_UA : " H |- ((UA usr r) && ((UP::'a
=> 'a wff) usr) && ((RP::'a => 'a wff) r))
              -> ( (PBOX ( (UA usr r) &&
(UP usr) && (RP r) ))
                 &&  (BOX  ( (UA usr r) &&
(UP usr) && (RP r) )) ) "
```

Since we want operations to be deterministic, the next axiom ensures the functional character of operations. This means, that the application of any operation to an arbitrary object in the domain of this operation can have one and only one definite result. In other words any operation will always deliver an unambiguous result when applied to an object in its domain, i.e. if the predicate *Eval* is true for the triples (x,y,z) and (x,y,w), then w and z have to be identical.

2.1.3. Functional character of operations

axioms

```
eval_ax:
"H |- AL1 x y. (Operation x && Object y ->
(AL1 z. (Object z && Eval x y z ->
(AL1 w. (Object w && Eval x y w ->
w === z)))))"
```

The following axioms are the actually important ones ensuring the RBAC properties. The axiom *spec1* says, that AUTH being true for a triple (usr,operat,obj) implies, that there is a role r having the permission to apply the operation *operat* to the object *obj* and the user *usr* has activated role r.

The axiom *spec2* demands, that EXEC(usr,operat,obj) implies AUTH(usr,operat,obj) (compare remarks at 2.1).

And finally *spec3* enforces, that ACTIVE_FOR(usr,r) at an arbitrary point of time implies, that UA(usr,r) is true at the present point of time. In the familiar notation of temporal logic spec3 might approximately look as follows:

$\forall usr, r :$
$((\diamondminus ACTIVE\_FOR(usr,r)) \vee (\Diamond ACTIVE\_FOR(usr,r)))$
$\Rightarrow UA(usr,r))$

2.1.4. Basic axioms

axioms

```
spec1 :        "H |- (AUTH usr operat obj)
-> (EX1 r. Role r && ((UA usr r) && ((PA r
operat obj) && (ACTIVE_FOR usr r))))"
spec2 :        "H |- (EXEC usr operat obj)
-> (AUTH usr operat obj) "
spec3 :        "H |-  ( (PDMD ((ACTIVE_FOR
usr r) && ((RP::'a => 'a wff) r) &&
((UP::'a => 'a wff) usr)))
                     || (DMD ((ACTIVE_FOR
usr r) && ((RP::'a => 'a wff) r) &&
((UP::'a => 'a wff) usr))) )
                     -> ( (UA usr r) && (RP
r) && (UP usr) ) "
```

end

The symbols RP and UP in spec3 denote rigid predicates as in the preceding axioms. Thus we have encoded RBAC in Isabelle/HOL based on the previously encoded theory LTL. Having done this we could then use the above RBAC theory (RBAC.thy at [1]) as formalism to express an RBAC security policy as a first example presented in the next section. A further example designed to demonstrate the first-order properties of the employed LTL will be introduced in the subsequent section.

3. Dual control as example

In this section we will briefly introduce dual control and functional separation of duty as two general principles for separation of duty (see [4]). Then we will give a compact introduction to the famous secret sharing scheme of Adi Shamir, that we will use to establish dual control in an IT-system with RBAC.

3.1. Dual control and functional separation

Dual control (cf [4]) is a well-known and frequently used separation of duty principle. It is applied to operations which are considered too critical to leave their control to one person or entity. Of course dual control as principle has already been known for a long time. Opening banks safes for example often requires the use of two ore more keys controlled by different persons at the same time.
Another old and well-known security measure is functional seperation (see also [4]), whose purpose is the protection processes consisting of at least two stages. This is achieved by making sure, that every stage is performed by another person or entity. Thus the process cannot be controlled by one person alone. This is a quite familiar principle for financial issues in the economy (cf [13]).

3.2. Shamir's secret sharing scheme

Let us assume a bank wants to implement dual control for a safe using an IT-system. The safe shall be opened by a combination, i.e. a secret number, generated by the system instead of a material key. To implement dual control the system applies the secret sharing scheme of Shamir [17] to the combination that opens the safe, i.e. it chooses a random 2 degree polynomial $q(x) = a_0 + a_1x + a_2x^2, a_2 \neq 0$ where a_0 is the combination for the safe. A share is then a pair $(x_i, q(x_i))$ with $i \in \mathbb{N}, 0 \leq i \leq n$. Furthermore $x_i \neq 0$ for all $0 \leq i \leq n$ and $x_i \neq x_j$ for all $0 \leq i, j \leq n$ with $i \neq j$ hold. Thus a person having 3 shares can compute the combination (a_0) for the safe by polynomial interpolation. On the other hand for any number c and 2 arbitrarily chosen shares $(y, q(y))$ and $(z, q(z))$ there is a 2 degree polynomial $p(x)$ such that $p(y) = q(y), p(z) = q(z)$ and $p(0) = c$. For further details see [17].

3.3. An example for dual control

The IT-system of the bank shall have RBAC and meet the following demands (*security properties*) :

Any user entitled to assume the bank role director shall be able to have two different shares but no more, any user entitled to assume the bank role cashier is allowed to have one share but no more (we assume, that no two users get the same share and no user gets the same share twice). Furthermore no user shall be able to get a share without being entitled to assume the role director or cashier.

Let k be the number of users entitled to assume the role director and l the number of users entitled to assume the role cashier, then our scenario is a (3,n) threshold scheme with $n = 2k + l$ according to the terminology introduced by Shamir in [17].
So it is assured by the properties of this scheme and RBAC, that opening the safe takes the cooperation of no less than two users of which at least one is assigned to the role director, unless one user or an adversary manages to guess or eavesdrop a sufficient number of shares or the combination itself. If three or more users cooperate, it is sufficient, that three of them are assigned to the role cashier.
The remaining part of this section introduces an RBAC security policy and gives then a *brief sketch* of our *Isabelle/HOL verified* proof, that this policy has the required security properties.

3.4. RBAC security policy

The security policy can be expressed using the RBAC theory given in this paper. It is encoded in Isabelle/HOL as follows:

theory *Example = RBAC + Adv_Op_Rules :*

3.4.1. Predicates for roles, sorts, etc.

consts
```
Director :: "'a => 'a wff"
Cashier  :: "'a => 'a wff"
Key      :: "'a => 'a wff"
Share    :: "'a => 'a wff"
GetShare :: "'a => 'a wff"
usr :: "'a"
```

3.4.2. The share predicates

consts
```
share_1 :: "'a => 'a => 'a wff"
share_2 :: "'a => 'a => 'a wff"
```

```
share_3 :: "'a => 'a => 'a wff"
```

axioms

```
SHARE_1_ax : " H |- AL1 x y. share_1 x y ->
(User x && Key y)"
SHARE_2_ax : " H |- AL1 x y. share_2 x y ->
(User x && Key y)"
SHARE_3_ax : " H |- AL1 x y. share_3 x y ->
(User x && Key y)"
usr_ax : "H |- (User usr)"
```

Similar to RBAC theory the roles director and cashier, the sorts key (key being a synonym for combination) and share and the operation for getting a share are defined by predicates of the same name (*Director, Cashier, Key, Share, GetShare*). We may assume, that the operation is associated with a state which is altered each time the operation is executed and prevents the result of the present execution from being equal to a share obtained from any previous execution. Additionally we introduce a constant *usr* of sort User (see user_ax). The predicate User is defined in 2.1.1.
The predicate *share_1* is true for a pair (u,k) if and only if the user u has (at least) one share of the combination (resp. key) k. The meaning of *share_2* resp. *share_3* is obvious.
The following (auxiliary) axioms should be self-explaining (except maybe the last two). AUX5c basically makes sure, that the operation for getting a share is unique. AUX6 enforces, that a share of a combination can only be computed by this operation. The symbol "<-->" denotes logical equivalence.

3.4.3. Connection with RBAC

axioms

```
AUX1:  "H |- Director x -> Role x"
AUX2:  "H |- Cashier x -> Role x"
AUX3:  "H |- Key x -> Object x"
AUX4:  "H |- Share x -> Key x"
AUX5:  "H |- GetShare x -> Operation x"
AUX5a: "H |- (GetShare x) -> ((DOMAIN x y)
<--> Key y)"
AUX5b: "H |- (GetShare x) -> ((RANGE x y)
<--> Share y)"
AUX5c: "H |- AL1 x y.(GetShare x) &&
(GetShare y) -> x === y"
AUX6:  "H |- Key y && Eval x y z ->
GetShare x && Share z"
```

Finally we mention some of the axioms to determine the temporal behaviour of the share predicates and the operation for getting shares, etc. For reasons of convenience we will present them in a notation more similar to that in literature. Therefore we replace *PNEXTE* by the symbol $\odot$, *BOX* by the symbol $\Box$ and *PDMD* by the symbol $\diamondsuit$. For this and other reasons our presentation may look quite different from that for Isabelle.

$\forall usr, k, op:$
$A1)\quad share\_1(usr,k) \Rightarrow \diamondsuit(\exists op1 : GetShare(op1) \wedge EXEC(usr,op,k) \wedge \odot(\neg share\_1(usr,k)))$

$A2)$
$(GetShare(op) \wedge EXEC(usr,op,k) \wedge \odot(\neg share\_1(usr,k))) \Rightarrow ((\Box share\_1(usr,k)) \wedge \neg share\_2(usr,k))$

$A3)\quad share\_2(usr,k) \Rightarrow$
$\diamondsuit(\exists op1 : GetShare(op1) \wedge EXEC(usr,op,k) \wedge \odot((\neg share\_2(usr,k)) \wedge share\_1(usr,k)))$

$\vdots$

Additionally we demand:

$B1)\quad \forall usr, op, k:$
$(GetShare(op) \wedge EXEC(usr,op,k)) \Rightarrow$
$(\forall u : (EXEC(u,op,k) \Rightarrow u = usr))$

$B2)\quad \forall op, obj1, obj2:$
$(Eval(op,obj1,obj2) \Rightarrow \neg Key(obj2))$

B1 assures that no two users can get a share for the same key at the same time. B2 excludes the existence of a user who can choose a combination for the safe, since this user then could open the safe alone.
Finally we introduce the essential authorisation constraint:

$C1)\quad \forall usr, op, k, c, d:$
$(AUTH(usr,op,k) \wedge GetShare(op)) \Leftrightarrow$

$(ACTIVE\_FOR(usr,c) \wedge Cashier(c) \wedge$
$\odot\,(\neg share\_1(usr,k))) \vee (ACTIVE\_FOR(usr,d)$
$\wedge\, Director(d) \wedge \odot(\neg share\_2(usr,k)))$

3.5. Security properties

Thus we have at last gathered the necessary constraints for our RBAC security policy. Many of those are rather technical, but necessary for a correct specification. So we are now in the position to deduce some properties for an arbitrarily chosen user (remind the previously defined constant *usr*).
Using the axioms for the share predicates (A1, etc.) we can deduce the following lemma:

lemma `share2_imp_1 : "H |- share_2 usr k -> share_1 usr k"`

Making use of C1 and the axioms for the share predicates we conclude:

lemma *share_3 : " H |- NOT (share_3 usr k) "*

Thus our security policy prevents any user from acquiring three shares within the system, which is one of our desired security properties (see 3.3)
In order to prove the remaining security properties we now combine lemma share2_imp_1, the axioms for the share predicates, the basic axioms of RBAC (spec1, spec2 and especially spec3) and the axiom C1, we can deduce the following two lemmata:
lemma *l1 : " H |- (share_1 usr k) -> (((UA usr c) && (Cashier c)) || ((UA usr d) && (Director d))) "*
lemma *l2 : " H |- (share_2 usr k) -> ((UA usr d) && (Director d)) "*

By the lemmata share_3, l1 and l2 the security properties defined in 3.3 are fulfilled by the given security policy. As already mentioned the proofs for these lemmata have been verified by Isabelle/HOL (cf Example.thy at [1]).

4. Why use a theorem prover?

Considering the somewhat tedious work of the last section the readers may wonder, if it would not have been easier to use model checking instead of a theorem prover. Of course model checking is known as a valiant method especially with respect to propositional LTL (cf [8]). Unfortunately first-order LTL as any first-order logic is generally undecidable and thus not so apt for model checking. Furthermore there are known security policies like operational dynamic separation of duty (see [12]) which cannot be expressed without first-order properties. Under these circumstances theorem proving may be a good option as illustrated in the following (theoretic) example.
Suppose we have a predicate *PREDEC* applicable to users indicating, that till the present time any operation which has been applied to an object by the user in question has also been applied to the same object by a different user coevally or at an earlier point of time:
$\forall usr, op, obj :$
$PREDEC(usr) \Leftrightarrow \boxminus(EXEC(usr, op, obj) \Rightarrow (\exists u : \diamondminus(\neg(u = usr) \wedge EXEC(u, op, obj))))$
Let there also be another predicate *ONLY* for users being equivalent to the statement, that there is an operation for whose application to any object till now *only* the considered user could have been authorised. Furthermore at the present time there is an object, such that the user is authorised to apply this operation to it:
$\forall usr, obj :$
$ONLY(usr) \Leftrightarrow \exists op : ((\forall u : \boxminus(AUTH(u, op, obj) \Rightarrow u = usr)) \wedge \exists obj1 : AUTH(usr, op, obj1))$

Suppose now, that the predicate PREDEC is true for a user and additonally every operation for which there is at least one object such that the user is authorised to apply this operation to it is in fact applied to some object by the user. The reader then (remembering spec2) may suspect, that in this case the predicate ONLY cannot be true for the same user, which is correct:
$\forall usr :$
$((\forall op : ((\exists obj : AUTH(usr, op, obj)) \Rightarrow (\exists obj1 : EXEC(usr, op, obj1)))) \wedge PREDEC(usr))$
$\Rightarrow \neg ONLY(usr)$
But proving this (again for an arbitrarily chosen user *usr*) by model checking seems quite difficult. On the other hand it can be done in first-order LTL. The authors have verified such a proof by Isabelle/HOL (cf Example2.thy at [1]).

5. Conclusion and future work

We have demonstrated, how a simple RBAC security policy based on first-order LTL can be encoded in Isabelle/HOL. Furthermore we have given proofs for some properties of this policy, that have been verified by Isabelle/HOL. In a second example we have demonstrated the advantages of a theorem prover for first-order LTL. The axiomatisation of the first-order LTL itself was not a topic of this paper and shall be described elsewhere.
For the future we intend to extend our temporal calculus to handle role hierarchies. Furthermore UA and PA shall become flexible predicates in order to allow the addition of administrative roles. In the long term we intend to employ also model checking for policy analysis especially with respect to consistency checking of security policies. We hope to find further inspiration with respect to consistency checking in the work of Bertino (cf [6]).
It remains future work to apply our calculus in the domain of clinical information systems in order to analyse real-world policies in an area with high security and privacy demands. Due to the fact that we consider a temporal calculus, safety properties (e.g., for railway control systems or in the avionics domain) can also be investigated.

References

[1] http://www.tzi.de/˜sohr/RBAC.

[2] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, Sept. 1993.

[3] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols : The SPI calculus. Technical Report UCAM-CL-TR-414, University of Cambridge, Computer Laboratory, Jan. 1997.

[4] R. J. Anderson. *Security Engineering — A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2001.

[5] G. Bella, F. Massacci, and L. C. Paulson. Verifying the SET purchase protocols. Technical Report UCAM-CL-TR-524, University of Cambridge, Computer Laboratory, Nov. 2001.

[6] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1):65–104, Feb. 1999.

[7] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report 39, DEC System Reearch Center, Feb. 1990.

[8] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[9] D. F. Ferraiolo, D. M. Gilbert, and N. Lynch. An examination of federal and commercial access control policy needs. In *Proc. 16th NIST-NCSC National Computer Security Conference*, pages 107–116, 1993.

[10] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramoli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.

[11] R. Goldblatt. *Logics of Time and Computation, Second Edition, Revised and Expanded*, volume 7 of *CSLI Lecture Notes*. CSLI, Stanford, 1992 (first edition 1987). Distributed by University of Chicago Press.

[12] T. Mossakowski, M. Drouineaud, and K. Sohr. A temporal-logic extension of role-based access control covering dynamic separation of duties. In *10th International Symposium on Temporal Representation and Reasoning / 4th International Conference on Temporal Logic (TIME-ICTL 2003), Cairns, Queensland, Australia*, July 8–10 2003.

[13] M. J. Nash and K. R. Poland. Some conundrums concerning separation of duty. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 201–207, 1990.

[14] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer Verlag, 2002.

[15] L. C. Paulson. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, Aug. 1999.

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

[17] A. Shamir. HOW TO SHARE A SECRET. Technical Memo MIT/LCS/TM-134, Massachusetts Institute of Technology, Laboratory for Computer Science, May 1979.

[18] P. Torrini. *Qualitative Spatial Reasoning with Super-Intuitionistic Logic*. PhD thesis, University of Leeds, UK, 2004.