Abstract—This paper extends RBAC sessions with share-
ability, reusability and switchability properties. We define
the Smatch (Secure MANagement of swiTCH) model in
which authorized users can join, leave, reopen and reuse
dynamic sessions. In Smatch, subjects can also share
sessions and dynamically switch their role or function with
other subjects from the same or different organizations.
Subjects can authenticate using their function which will
automatically activate the set of roles associated with this
function. The Smatch model is based on first order logic
with actions. It provides means to specify contextual access
control and authentication policies which apply to control
functional behavior of dynamic sessions. We suggest an
implementation of Smatch using virtual machines.

Keywords—OrBAC; Session; Switch; Virtualization

I. Introduction

Traditional access control models are usually con-
strained by two main assumptions. The first assump-
tion is related to the authentication part of the access
control. It is assumed that subjects are previously au-
thenticated before applying access control policies, but
there are no means to explicitly specify how subjects
are actually authenticated (password authentication,
strong authentication, one time password, SSO, etc.).
The second assumption, which is rather related to
the most widely used access control model RBAC
and those models extending it, stipulates that once
authenticated, subjects must create a session to activate
their roles. This is necessary to activate permissions
assigned to the role. But, again, there are no means
in these models to specify how a session is created
or deleted and no means to specify how roles are
activated or deactivated. Moreover, a session cannot be
shared by several subjects and a session, once closed,
cannot be reopened by the same user, or a user having
the same profile and the appropriate authorizations.

However, these assumptions are no longer appro-
priate to manage security in new frameworks based
on virtualization. Virtualization provides means to
support new behaviors. For instance, we shall show
how to adapt the VMotion environment provided by
VMWare so that a user can create a session, start some
work and then save the session. It is also possible for
the same or another user to restore the session in an-
other context, for instance another computer. “Session
reusability” is useful in many situations, like continuity
of services or tasks, management of urgency, etc.

In this paper, we aim at extending session man-
agement suggested in RBAC to securely handle such
session reusability. The contributions of this paper are
twofold: (1) making explicit the authentication policy
specification so that we offer the possibility to combine
authentication requirements with access and usage
control requirements, (2) extending specification and
management of a dynamic session to enable:

- Shareability: more than one user or role can use
the same session, making it possible in that way
collaborative works within the same session,
- Reusability: a user can close its current session
and reopen it while keeping the same states and
environments (running applications, opened files,
connections that were in progress, etc. when the
session has been interrupted or closed).
- Switchability: it must be possible, in some circum-
stances, for any user of the system undertaking
some tasks to transfer the deal to another user
and so let him use or recover the same session.
This user which takes up again the session may
be in a different time space or localization than
the original or the predecessor user of the session.

The model, called Smatch for Secure MANagement
of swiTCH, we define to securely manage a dynamic
session has two parts: a functional part and a security
part. The functional part specifies the different opera-
tions users can execute on a dynamic session including
creation, joining after authentication, various types of
switch, etc. The security part is defined as an extended
RBAC model. It provides means to specify contextual
authorization and authentication policies associated
with sharable and reusable sessions.

From an implementation point of view, the func-
tional part of this model is enforceable using virtual
machines. The security part is implemented using the
four modules: authentication manager, session monitoring, context handler and authorization manager.

This paper is organized as the following. Section 2 presents a use case which illustrates different possible applications of our approach. Section 3 investigates related works showing limitations of previous approaches and the originality of the issues addressed in this paper. Section 4 formally defines the Smatch model and shows how it applies to control behavior of dynamic sessions. Section 5 explains the implementation of the Smatch model based on virtual machines. Finally, section 6 concludes the paper.

II. Motivating Example

We present here a simplified scenario related to some public safety use case. The objective is to highlight the importance to exchange, beyond data, ongoing processes and working environments. That goes without saying security must be specified and implemented to provide trust in the global system. Let us consider a security and public safety center. This police headquarter houses several PCs that centralize images taken by video monitoring system of the road traffic, cameras of corridor of buses, ATM, back streets, and so on, and information gathered from different sources about suspicious events, profiles of criminals, juvenile offenders or children in conflict with the law, and so on. Also, some specialized applications run on some of these PCs, used for instance to merge, aggregate and correlate information or check the veracity of alarms.

Eugene, one of the policemen in the security and public safety center, initiates a session on a PC. Let us consider that some road accident occurs and the responsible eloped. Alarms are raised and handled by the personnel of the center. In his current session, Eugene launches several applications to gather and analyze information related to the event that triggered the alarms. He also opens some pre-formatted files to fill in them with useful data needed to conduct the investigation to identify, pursue and catch the fugitive. Afterwards, Eugene has to move to the location of the accident. He must recover the session initiated in the center from the PC integrated in his police car while preserving the same working environment to waste no time. This corresponds to a switch of location.

After the police investigation, some information must be sent to a safety center in another country; the car of the fugitive has a foreign country registration number. The policeman Philippe, Eugene’s colleague, sends the useful information to this center, using the PC integrated in Eugene’s police car and makes use of the applications and files already initialized and filled by Eugene. This corresponds to a switch of roles. Although he uses Eugene’s working environment, Philippe has to log in, without creating a new account, allowing the system to manage his authorizations. The sent information will be handled by another organization, that is a switch of organization.

We can obviously go further in the refinement and unfolding of this scenario to catch other kinds of switch. The main concept to consider, to deal with our scenario, is the dynamic transfer. For this purpose, we study the security policies that allow us to specify the notion of “switch”. We give in Figure 1 the different kinds of switch we can consider.

![Figure 1. Kinds of “Switch”](image)

**Role switch.** It allows a user to hand over to another user in order to continue the fulfillment of a given task by preserving his/her working environment.

**Activity switch.** It allows to hand over to another user in order to fulfill another activity by preserving the working environment of the original activity.

**Context switch.** It corresponds to a transfer without preserving completely the initial working environment. For instance, a user can make a transfer to another user, but the initial context is modified during the transfer, for instance to preserve the user privacy.

**Organization switch:** The user hands over to another user belonging to another organization. This is probably the most complex case since the user who is targeted by the switch, will not be, probably, assigned to the “same” role as the original user. Similarly for the context.

Note that to deal with some scenarios, it must be possible to combine different kinds of switch. Namely, the organization switch is generally accompanied by a role or/and a context switch.

III. Related Work

In RBAC [1], roles are assigned to users and permissions are assigned to roles. Roles are structured hierarchically and permissions are inherited though
models have been proposed. In [15] a context-sensitive access control many context-sensitive access control the computing domain. In order to enable fine grained that are relevant for performing appropriate actions in
lates to the characterization of environment conditions
an organization structure and hierarchy in workflow
ieties and appropriate relations. The importance of using
RBAC extended with case and organization unit enti-
propose a workflow access control model based on
through a service as an interface. Wainer et al. [12]
binds them than associating roles with tasks, SOWAC binds them
(SOWAC) model [11], service is the abstraction of a
access control configuration will not result in the
leakage of a right to an unauthorized principal. A
RBAC based workflow access control model in which
tasks and permissions are assigned to the roles is also
presented in [6].

Shafiq et al. [7] present an adaptive real-time workflow-based collaborative system. They use the
Generalized Temporal Role-Based Access Control (GTRBAC) model [8] to capture the real-time dependencies of such workflow applications. Russello et al. [9] propose the Workflow-Based Access Control (WBAC) model, an access control mechanism that adapts the access rights of subjects to the actual tasks that they have to fulfill. Importance of time in workflow access control models is also discussed in [10].

In the Service-Oriented Workflow Access Control (SOWAC) model [11], service is the abstraction of a task and the unit for applying access control. Rather than associating roles with tasks, SOWAC binds them through a service as an interface. Wainer et al. [12] propose a workflow access control model based on RBAC extended with case and organization unit entities and appropriate relations. The importance of using an organization structure and hierarchy in workflow access control models is also noticed in [13].

Various definitions of context have been proposed, see for example [14]. Broadly, the notion of context relates to the characterization of environment conditions that are relevant for performing appropriate actions in the computing domain. In order to enable fine grained access control many context-sensitive access control models have been proposed. In [15] a context-sensitive access control model is defined that includes a context model, a context-sensitive policy model and a context-sensitive request model. Covington et al. [16] introduce the notion of environmental role, and provide a uniform access control framework that can be used to secure context-sensitive applications. Georgiadis et al. [17] discuss the integration of contextual information with team-based and role-based access control. Several authors, for example [18], [19], propose a context-based access control model for web services.

By analyzing the previously mentioned access control models for workflows, we noticed that most of them are based on the RBAC model which is extended with context awareness and some other concepts specific for business processes. The approach we suggest in this paper is based on several new concepts which are significantly different from previous proposals:

**Dynamic session.** As suggested in the introduction, our concept of dynamic session aims at satisfying the shareability, reusability and switchability properties. None of these properties are provided by the RBAC sessions. In our approach, a user may create a dynamic session and starts the execution of a task, suspend the session, reopen it, and continue the execution of this task in another context. Also, using the switch operation, this task may be continued by another user. None of these dynamic behaviours can be modelled in TBAC or TRBAC. Even if dynamic sessions may be shared by several users, there is generally no pre-defined workflow associated with a dynamic session. Thus, the security issues raised by dynamic sessions are definitely different from the ones addressed by workflow management.

**Dynamic function.** To ease the switch operation from some user who starts a given task to another user who will continue the task, we introduce the concept of function authentication. When function authentication is used, the user does not provide her identity but her function. If the authentication is successful, then a set of roles associated with her function is automatically activated. If this user then switch to another user and this latter user is authenticated through the same function, then this user will reopen the same environment with the same function, but possibly with different
permissions since the context has changed.

Notice that this process of switch is different from delegation. If the switch operation is replaced by delegation, then the working context of the initial session will be lost when the delegated user will activate the delegated function (or role). This is typically what we want to avoid when using the switch operation.

IV. THE SMATCH MODEL

We can now present the Smatch (Secure MANagement of swITCCH) model. This model is based on first order logic with action. First order logic is used to represent the system state at a given time. A system state is represented by a set of ground facts and a set of derivation rules having the form \( P_1 \land \ldots \land P_n \rightarrow P \).

We assume that the derivation rules are syntactically compatible with Datalog [20]. As usual, we assume that negative literals are allowed if it is possible to stratify the derivation rules. Stratifying a Datalog program consists in ordering derivation rules so that if a rule contains a negative literal then the rule that defines this literal is computed first. A stratified Datalog program with negation is computable in polynomial time through the computation of a fix point.

We also assume that the system state may include a set of constraint rules. A constraint rule is a derivation rule whose conclusion is the predicate \texttt{error}. If it is possible to derive \texttt{error}, then we say that the system state is inconsistent. In the following, we shall assume that the initial system state is always consistent.

Starting from the initial state, the system state can then change due to the execution of actions. Actions are specified through dynamic effect laws having the following form: \( A(s,o) \) causes \( P \) if \( Q_1 \land \ldots \land Q_n \)

\( \rightarrow \) P.

where \( A(s,o) \) represents the execution of action \( A \) by subject \( s \) on object \( o \) and \( P,Q_1,\ldots,Q_n \) are negated or unnegated application-dependent predicates. When \( P \) is unnegated (resp. negated), the rule specifies that the occurrence of \( A(s,o) \) makes \( P \) true (resp. false) if the conditions \( Q_1,\ldots,Q_n \) are true when \( A(s,o) \) occurs. To avoid ambiguities when a negated predicate is used in the effect of an action\(^1\), we consider that there are two types of predicates: ground predicates and derived predicates. A ground predicate cannot appear in the conclusion of a derivation rule. A derived predicate cannot be used in the effect of an action.

If none of the conditions of some effect laws of action \( A(s,o) \) is true then execution of action fails, i.e. the system remains unchanged. Also, if after the execution of some action \( A(s,o) \), the system state is inconsistent, then we assume that the action fails. Due to this last assumption, the system state always remain consistent if the initial state is consistent. If the execution of some action \( A(s,o) \) does not fail, then we assume that the predicate \( Do(s,A,o) \) is true in the state resulting from the execution of action \( A(s,o) \).

As explained in the introduction, the Smatch model has actually two parts: A functional one, which specifies functional behavior of dynamic sessions through specification of action effect laws, and a security one, which controls access to dynamic sessions. We now present the different parts of the Smatch model.

A. Access control policy specification

The Smatch access control model is defined as an extension of the Organization-Based Access Control model (OrBAC) [21]. OrBAC natively provides several concepts we need to specify the Smatch model. Traditional models are based on subjects that have the right to make actions on objects. An abstraction level is offered by OrBAC to categorize subjects into role, objects into view and actions into activity. This abstract level is introduced to design implementation-independent policies. These abstract entities are designed within an organization to provide interoperability between organizations and enforce separation of duties.

The organization is the central concept of OrBAC. In OrBAC, several organizations may specify their own security policy. An organization groups a set of subjects and is in charge of defining and enforcing the policy applied to these subjects when they perform actions on objects. So, the specification of the policy is parameterized by the organization. This is used to handle simultaneously several security policies associated with different organizations. Organizations may be structured hierarchically so that the policy of a given organization is inherited by its sub-organizations. We shall use this possibility to consider that a dynamic session is actually a (virtual) sub-organization of the organization where this session is created.

In OrBAC, a security policy corresponds to a set of contextual organization Privileges. Abstract organization privileges, such as permission, are expressed through the predicate: \( \text{Permission} (\text{organization}, \text{role}, \text{activity}, \text{view}, \text{context}) \)

OrBAC uses three kinds of privileges: permission, prohibition and obligation. Detection and management between these different kinds of conflicts is discussed in [22]. However, for the sake of simplicity, we shall only consider permissions in this paper.

A permission corresponds to a relation between roles, views and activities at the organizational level. The concrete policy is logically derived from abstract privileges, according to derivation rules. The corresponding derived concrete permission is \texttt{Is permitted}. 

\(^1\)It there is a derivation rule \( P_1 \land \ldots \land P_n \rightarrow P \), then an ambiguity exists when an action whose effect is \( \sim P \) is executed since it is not clear which of the \( P_1,\ldots,P_n \) should be removed from the system state.
It computes if a given subject, belonging to a role, can perform a given action, belonging to an activity, on a given object, belonging to a view.

The relationship between abstract entities and concrete entities is modeled using three built-in predicates, which specify conditions over subjects' domain $S$, actions' domain $A$ and objects' domain $O$. Let $\text{Org}$ be the set of organizations:

- $\text{Empower}(\text{org}, s, r)$ means that $s$ plays the role $r$ within $\text{org}$.
- $\text{Consider}(\text{org}, \alpha, a)$ means that $\text{org}$ considers that $\alpha$ is implementing the activity $a$,
- $\text{Use}(\text{org}, o, v)$ means that $o$ uses the object $v$ in the view $v$.

Notice that $\text{Empower}$ is used to specify that a role is active for this subject. In the Smatch model, we have to make a distinction between role assignment and role empowerment. For this purpose, we extend the OrBAC model with the following $\text{assign}$ predicate:

- $\text{Assign}(\text{org}, s, r)$: means that subject $s$ is assigned to role $r$ in organization $\text{org}$, but $r$ may be not active.

Of course a subject cannot be empowered in a given role without being assign to this role: $\text{Empower}(\text{org}, s, r) \land \neg\text{Assign}(\text{org}, s, r) \rightarrow \text{error}$

Contexts are designed to handle dynamic parameters of a policy. A context is defined as an abstract condition that takes into account such environment parameters when specifying abstract organization privileges. They are designed to allow the definition of a dynamic security policy. Contexts are constraints that model extra conditions a subject, an action and an object must satisfy to activate a privilege. An OrBAC built-in predicate $\text{Hold}$ permits linking those entities:

- $\text{Hold}(\text{org}, s, a, o, c)$ means that context $c$ holds between subject $s$, action $a$ and object $o$ within $\text{org}$.

The OrBAC model defines five types of contexts [23]:

1. spatial context that depends on the subject position,
2. temporal context that depends on the time of the subject request,
3. user-declared context that depends on parameters declared by the subject, (4) prerequisite context that depends on a relation between the subject, the action and the object, and (5) provisional context that depends on the previous actions of the subject. We also call $\text{Default}$ the context which is always true.

Using these materials, concrete permission derivation, is modeled by the following derivation rule:

$\text{Permission}(\text{org}, r, a, v, c) \land \text{Empower}(\text{org}, s, r) \land \text{Use}(\text{org}, o, v) \land \text{Consider}(\text{org}, \alpha, a) \land \text{Hold}(\text{org}, s, a, o, c) \rightarrow \text{Is\_permitted}(s, a, o)$

That means $s$ is permitted to perform $a$ on $o$, if (1) there is an abstract permission in organization $\text{org}$, which allows the role $r$ to perform activity $a$ on view $v$ within the context $c$, and (2) $\text{org}$ empowers subject $s$ in role $r$ and (3) $\text{org}$ uses object $o$ in view $v$ and (4) $\text{org}$ considers that action $a$ implements activity $a$ and (5) within $\text{org}$, the context $c$ holds between $s$, $a$ and $o$.

Access control is then defined by the following constraint: $\text{Do}(s, a, o) \land \neg\text{Is\_permitted}(s, a, o) \rightarrow \text{error}$

This constraint says that if a subject executes an action whereas this subject is not permitted to do so, then an error occurs. Thus the execution of this action fails and the system state remains unchanged.

To structure the set of entities and authorizations in OrBAC, hierarchies and inheritance mechanisms are introduced. Roles, views, activities, contexts and organizations may be structured according to hierarchies. In this case, privileges are inherited through this hierarchy. For this purpose, the predicate $\text{Sub}\_\text{role}$ is used to specify role hierarchies. We assume that $\text{User}$ represents the most general role in the role hierarchy. Thus every role inherits from the role $\text{User}$. Similar predicates for activity (Sub\_activity), view (Sub\_view), context (Sub\_context) and organization (Sub\_organisation) are defined for hierarchies.

In the following, we consider that a subject can only get a permission if this permission is assigned to a role and this role is assigned to this subject. However, this is not sufficient: The role must be active, i.e. the subject must be empowered in the role. As a consequence, a subject who is not initially empowered in any role will never be able to (legally) execute any action in the system. To solve this problem, we shall assume that the $\text{User}$ role does not need to be activated. This is modeled by the following derivation rule:

$\text{Assign}(\text{org}, s, \text{User}) \rightarrow \text{Empower}(\text{org}, s, \text{User})$

Notice that this issue is not addressed in the RBAC model because it does not explicitly model role activation (as well as authentication and session creation).

B. Authentication Policy Specification

The second part of the Smatch model corresponds to authentication policy specification. Since access control is commonly based on the identity of the subject who requests an access, authentication is necessary, though not sufficient, to enforce security. Authentication actions may be implemented using credentials, smart cards, a public key infrastructure and so on. Furthermore, tokens may be assigned to users, to track their authentication state which helps systems manage authorizations without frequently asking for new authentication actions. We formalize these actions like the following using dynamic effect laws.

- **Password Authentication**: this is the most commonly used authentication which is modeled by the action, $\text{password\_authentication}(s, \text{pass}, \text{org})$ that means subject $s$ uses password pass to authenticate itself in organization $\text{org}$. We define it as follows,
Action passwd_authentication(s, pass, org)
Causes authenticated_by_password(org, s)
If password(org, s, pass)
i.e. if subject s provides its password pass in organization org (represented by predicate password(org, s, pass)), then s will be authenticated by password in organization org.

- **Strong Authentication**: we consider an action, strong_authentication(s, pass, tok, org) that means subject s uses password pass and token tok to authenticate in organization org.
  Action strong_authentication(s, pass, tok, org)
  Causes strongly_authenticated(org, s)
  If password(org, s, pass), token(org, s, tok)

We also consider that subject may be authenticated in accordance with the function she has to perform in the system. This means that this subject will not provide her identity to authenticate but her function.

- **Function Authentication**: we consider an action, function_authentication(s, f, pass, org) that means pass is the password associated with function f^2 that subject s uses to authenticate in organization org that we define as follows,
  Action function_authentication(s, f, pass, org)
  Causes function_authenticated(org, s, f)
  If password(org, f, pass)

Authentication through a function will automatically activate a set of roles. For this purpose, we generalize the assignment predicate to make it possible, as is the case for subject, to assign roles to function: assign(org, f, r) means role r is assigned to function f in organization org.

In Smatch, the execution of any action must be permitted by the access control policy. This applies to authentication as well. For example, one may specify that, in organization org, any subject empowered in the role User is permitted to authenticate by password: Permission(org, User, passwd_authentication, Password, Default). However, execution of other means of authentication may require more restricted privileges.

C. **Dynamic session management**

In Smatch, we give a modeling of a dynamic session, its creation and structuring, before specifying how roles are activated and deactivated within a session. This corresponds to the functional part of Smatch.

1) **Session representation**: We need first to designate the considered session. For this purpose, we define a special view session and use(org, ss, session) means that organization org uses object ss as a session. A session is normally initiated by some subject: session_initiator(ss, subj). And a state (active or idle) is associated with a session: session_state(ss, state).

2) **Session Creation**: We consider an action create_session(subj, ss, org) which we formally define using the following action specification rule,

  Action create_session(subj, ss, org)
  Causes use(org, ss, session) ∧ session_initiator(ss, subj) ∧ session_state(ss, active)
  If True.

This is the functional specification of session creation. Of course, the access control policy must specify which subject is permitted to create sessions. For instance, one may consider that, once authenticated, every user is permitted to create sessions:

Permission(org, authent_user, create, session, Default)
and authenticated(org, s) ∧ empower(org, s, User) → empower(org, s, authent_user)

where authenticated(org, s) is true if subject s is authenticated in organization org (by password, strong authentication or function).

Once created, a session becomes a (virtual) suborganization of the organization in which it was created: use(org, ss, session) → sub_organization(ss, org)

This means that the session inherits every permission from the parent organization. We assume that inheritance also applies to the assign, use, consider and hold predicates.

3) **Session Management**: The dynamic aspect of a session can be captured by four states and seven transitions depicted in the state diagram of Fig.2. The operational semantics is given below using dynamic effect laws.

**Joining a session.** A subject may ask to join an active session. It is stated formally,

  Action join_session(subj, ss)
  Causes session_member(ss, subj)
  If session_state(ss, active).

**Leaving a session.** A subject may ask to leave an active session. We specify it formally as the following,

  Action leave_session(subj, ss)
  Causes ¬session_member(ss, subj)
  If session_state(ss, active) ∧ session_member(ss, subj).

**Sharing a session.** A subject may ask to share an active session with another organization,

  Action share_session(org, subj, ss)
  Causes use(org, ss, session)
  If session_state(ss, active).

**Unsharing a session.** A subject may ask to unshare an active session with another organization, formally:

  Action unshare_session(org, subj, ss)
  Causes ¬use(org, ss, session)
  If session_state(ss, active) ∧ use(org, ss, session).

**Asleeping a session.** A subject may ask to asleep an active session:

2We could also define strong_function_authentication if function authentication requires both a password and a token...
Awaking a session. A subject may ask to awake an idle session, formally stated:

**Action awake_session(subj, ss)**

**Causes** session_state(ss, active)

If session_state(ss, idle).

Access control policy must specify which roles are permitted to manage session, i.e. who is permitted (may be contextually) to join, leave, share, unshare, asleep, awake and delete a session.

Furthermore, it might be possible to consider different kinds of sessions. We define thus a predicate session_type(ss, type) where type can be consultation, rescue, emergency and so on.

4) Role Activation. Once the session is created, a subject who is member of the session can activate a role. Let us consider an action activate_role(subj, role, ss) that means subject subj activates role r in session ss. We define it formally as following,

**Action activate_role(subj, role, ss)**

**Causes** empower(ss, subj, role)

If session_member(ss, subj).

In the organization, an authenticated user is permitted to activate one of its assigned roles:

\[ \text{Permission}(org, \text{authent_user}, \text{activate}, \text{role}, \text{assigned_role}). \]

and assign(org, S, R) → hold(org, S, R, assigned_role)

This permission and this hold definition are inherited in the active session as previously specified.

5) Role Deactivation. Once the session is created, a subject who is member of the session can deactivate a role. Let us consider deactivate_role(subj, role, ss) an action that means subject subj deactivates role r in session ss. Formally stated,

**Action deactivate_role(subj, role, ss)**

**Causes** ¬ empower(ss, subj, role)

If session_member(ss, subj).

An authenticated user is permitted to deactivate one of its active role, and the inheritance rules apply for active session like for the role activation case:

\[ \text{Permission}(org, \text{authent_user}, \text{deactivate}, \text{role}, \text{active_role}). \]

and empower(org, S, R) → hold(org, S, R, active_role)

D. Switch Modeling

Based on the material presented in the previous sections, it is now possible to model and enforce switching cases of user, of role and of organization.

Switching a user occurs when a user usr, leaves the session and is replaced by another user subj who joins the same session. subj may join the session after a login/password authentication. Thus, although subj joins the same session as usr who leaves it, subj according to the authorization policy, may have less, more or the same privileges as usr. The leaving user loses privileges she had when she was a member of the session:

**Action user_switch(subj, usr, ss)**

**Causes** empower(ss, subj, r) ∧ ¬empower(ss, usr, r)

If session_state(ss, active) ∧ leave_session(usr, ss) ∧ empower(ss, usr, r) ∧ join_session(subj, ss) ∧ passwd_authentication(subj, pass, ss)

∧ assign(org, subj, r) ∧ sub_organization(org, ss)

In this user switch, the joining user has the same privileges as the leaving one. From the enforcement point of view, the working environment is maintained. For instance, all the processes that was running when usr lefted the session still carried on and subj can use the resulting data. Moreover, subj must be in this case a user known by the system, that is subj has an account and a login/password. This can harm the continuity of service if there is a need to create a new account and validate this action through an administration approval process.

We can weaken this user account creation constraint. Actually, subj can also join the same session with the same profile as the leaving user. The joining user has to perform function authentication and receives the same privileges as the leaving user. In this case, it is no
longer necessary to create an account for this user to join the session:

**Action** user_switch(subj, usr, ss)

**Causes** empower(ss, subj, r) \(\land\) ~empower(ss, usr, r)

If \(\text{session} \_\text{state}(ss, \text{active}) \land \text{leave} \_\text{session}(usr, ss) \land\)

\(\text{empower}(ss, usr, r) \land \text{join} \_\text{session}(subj, ss) \land\)

function\_authentication(subj, f, pass, ss) \(\land\) assign(ss, f, r)

However, we seemingly lose means to ensure accountability; the authentication is not user identity-based. A virtual identity may be assigned to each user login to the active session using a function authentication. Thus, correlating organizational and technical log auditing using virtual identities allows us to satisfy accountability requirement in this case.

**Switching a role** occurs, during an active session, when a user deactivates its current role \(R_r\) and activates another role \(R_{r'}\). So, he or she loses the privileges assigned to \(R_r\) and receives new ones provided by \(R_{r'}\):

**Action** role_switch(usr, \(r_r, r_{r'}, ss)\)

**Causes** empower(ss, usr, \(r_r\)) \(\land\) ~empower(ss, usr, \(r_{r'}\)

If \(\text{session} \_\text{state}(ss, \text{active}) \land \text{empower}(ss, usr, r_{r'}) \land\)

assign(ss, usr, \(r_{r'}\))

The working environment changes accordingly. For instance some data may become no longer accessible or some processes may be stopped. In the latter case, new activities may be started, and thus a switch of activity follows a switch of role.

**Switching an organization** occurs when a new organization shares the session and an existing one unshares this session.

**Action** organization\_switch(org_{ss}, org_{usr}, ss)

**Causes** use(org_{usr}, ss, session) \(\land\) ~use(org_{usr}, ss, session)

If \(\text{use}(org_{usr}, ss, \text{session}) \land \text{session} \_\text{state}(ss, \text{active})\)

We must highlight again here that the switch paradigm is under the control of three security axis, that is the switch beneficiary can evolve only in the sphere of authorizations bounded by these axis (Figure 3): (1) authentication mechanisms and policy, (2) access and usage control policy through the activation/deactivation of roles and contexts and (3) security administration which encompasses in particular joining, leaving, shareability, unshareability, activation and deactivation of sessions management.

**V. Switch Enforcement**

Smatch implementation is based on virtualization techniques – virtual machines and paravirtualization for setting up these machines in more powerful and simpler way. Managing the switch is reduced to a secure access and usage management of virtual machines. To remotely access a virtual machine, we chose TeamViewer\(^3\), a software package for remote control, desktop sharing, and file transfer between computers. It ensures a secure access using RSA private/public key exchange and AES session encoding. To enforce high availability of network machines, we deploy several servers with at least 2 GB of RAM, each hosting a set of users’ virtual machines. These servers perform load balancing and migration of machines when needed to satisfy availability requirements. We use VMware vSphere\(^4\) to implement our solution. There are several reasons that justify this technical choice: it supports all types of Linux and Windows virtual machines, it controls the creation of virtual network, it can manage the virtual environment using the web browser, live migration of virtual machines is possible with VMware VMotion\(^5\), it has a very active community, etc.

Back to our motivating example, let us take the case where the user has to move from his office and continue his work in another location where Internet connections are not available. Thus, he cannot have access to his machine remotely and reuse the ongoing active session. This is a switch of context; the location has changed. To restore the session in the new location, a first solution can be: when the user is about to move, he makes a copy of his virtual machine on a storage media. But this solution may be time consuming and is thus not acceptable in emergency situations. The second solution is to open a session and work directly with a virtual machine from a removable storage media like a USB key or an external hard driver. Based on comparison results we have performed of virtualization tools allowing users to use a machine from an external media, VirtualBox is faster in reading and writing respectively than VMware when loading a virtual machine from a storage medium ((6.07, 4.68) versus (8.65, 14.65)). But in the case of a remote access to the virtual machine via the network, the loading capacity of VirtualBox and VMware are almost the same in terms of loading and downloading speed ((5.2, 4.9) versus (4.1, 5.3)).

The aforementioned techniques allow the user to reuse his session every time and everywhere. But this is not completely satisfactory as he needs also to recover the contents of his session and its environment settings before he closed it. An open session contains a set of components such as running applications and files in use or browsers with a set of opened tabs. A recovered session must contain all the elements that were in use before the switch. Some of these elements may be recovered directly using the application itself if it can restore its previous state. For instance, generally for the web browser like firefox, a backup file of the last

\(^3\)http://www.teamviewer.com/

\(^4\)http://www.vmware.com/

\(^5\)http://www.vmware.com/products/vmotion/
visit of the browser and the various tabs is kept open under "/home/user/.firefox/". In this hidden folder, a set of files enables the browser to restore its previous state. In the case of files that were under processing in the closed session, the goal is to recover and automatically open all of them in the recovering session, in the same states as those when the session has been temporarily closed (location on the screen page being edited, cursor positioning on this page, maintaining the same changes,...). To do so, we developed a script that will be executed upon a closing of the session to recover the list of these files and another script is executed to open all these files when starting the new session that recover the closed one.

![Figure 4. Access and Usage management](image)

Opening files is preceded by a call to the security attributes manager module and the security policy server (Figure 4). The former is used to extract metadata. These data provide the descriptive and technical information about the considered files (type of document, user ID, users roles, active roles, address of the security server, public chapters of the document, confidential chapters, access or usage context, etc.) (see Figure 5). We use MotOrBAC [24], a support tool of the OrBAC model to implement this security server.

![Figure 5. Extracting Metadata](image)

According to the user’s authentication actions to access or join the session, the type of session, the environments and the metadata sent by security attributes manager, the server consults the security policy and takes decision (based on the metadata of Figure 5, the user who logged on with a role “Employee” will have access to a file containing only public chapters of the original file because the user’s role does not allow him to have an access to confidential content see Figure 6). Running applications are managed in the same way. They can be restarted or stopped according to the security server decisions. The contextual aspects (triggered events) are mutually managed by the security server and the security attributes manager.

![Figure 6. The resulting file after security server decision](image)

VI. Conclusion

In this paper we formally specify and implement a model called Smatch which provides more flexible session management than the one defined in traditional RBAC models. In particular, it is possible to specify shareability, reusability and switchability of sessions using Smatch. To reach this goal, we flatten the authentication specification and its links with access policy specification. Since a session is modeled as a dynamic sub-organization, Smatch is defined as an extension of the OrBAC model which makes explicit the organization dimension. We also show that the model can technically be enforced using emergent virtualization technologies. Due to space limitation, several related aspects in the context of session supplied with the aforementioned properties, have not been presented in this paper, like managing conflicts, contrary to duty or delegation, though well handled by the Smatch model. Among the works in progress that refine the model, we can mention the automatic generation of smart interfaces which are access and usage policy compliant when switches occur [25].

Acknowledgment

This work is part of the ITEA 2 Role-ID project. The authors would like to thank the DGCIS for its funding.

References


