Formal Verification of Liferay RBAC

Stefano Calzavara, Alvise Rabitti, and Michele Bugliesi

Università Ca' Foscari Venezia

Abstract. Liferay is the leading opensource portal for the enterprise, implementing a role-based access control (RBAC) mechanism for user and content management. Despite its critical importance, however, the access control system implemented in Liferay is poorly documented and lacks automated tools to assist portal administrators in configuring it correctly. To make matters worse, although strongly based on the RBAC model and named around it, the access control mechanism implemented in Liferay has a number of unconventional features, which significantly complicate verification. In this paper we introduce a formal semantics for Liferay RBAC and we propose a verification technique based on abstract model-checking, discussing sufficient conditions for the soundness and the completeness of the analysis. We then present a tool, called LifeRBAC, which implements our theory to verify the security of real Liferay portals. We show that the tool is effective at proving the absence of security flaws, while efficient enough to be of practical use.

1 Introduction

Liferay[1] is the leading opensource portal for the enterprise, adopted by important companies like Allianz, Cisco, Lufthansa and Vodafone, just to name a few [17]. Liferay allows portal administrators to conveniently manage both users and contents in a unified web framework. Users are typically structured into a hierarchy of *organizations*, where members of a child organization are also members of the parent organization. Contents, instead, are collected into *sites*, built as assemblies of different pages, portlets and social collaboration tools, like blogs and wikis. Both organizations and sites belong to a top-level *company*, and a single portal may host different companies at the same time.

For enterprises, the Liferay portal is at the core of the business process, since security-critical portlets may allow, for instance, to access sensitive information and/or to reorganize workflows. To ensure that private contents are only accessed by the intended recipients and that business processes are only handled by authorized users, Liferay implements a *role-based access control* (RBAC) mechanism.

1.1 Liferay RBAC

In the standard RBAC model, permissions are assigned to a relatively small and fixed set of roles, while roles are assigned to a potentially large and dynamic set

[1] `http://www.liferay.com`

of users: since user privileges only depend on the assigned roles, this approach simplifies the access control management task [7]. When role administration is itself role-based, like in the case of Liferay, the RBAC model is typically called *administrative* and abbreviated as ARBAC [20]. For the sake of simplicity and for consistency with the Liferay documentation, in this paper we uniform the two models and we just use the acronym RBAC everywhere.

Though extremely popular and widely deployed, real-world RBAC systems are notoriously difficult to get right, since the set of roles dynamically assignable to each user is easily under-estimated and occasional changes to the access control policy may introduce overlooked security flaws. The research community has then proposed formal methods as an effective tool to strengthen complex RBAC systems and ensure that they meet their intended goals [3, 9, 8, 2, 19]. Notable examples of useful security goals include role (un)reachability, ensuring that a given role granting powerful privileges is never assigned to untrusted users, or mutual exclusion properties, preventing the assignment of dangerous combinations of permissions to the same user.

Despite its critical importance and these well-known problems, the access control system implemented in Liferay is poorly documented and lacks automated tools to assist portal administrators in configuring it correctly. To make matters worse, although strongly based on the RBAC model and named around it, the access control mechanism implemented in Liferay does *not* constitute, strictly speaking, an RBAC system. First, users of the portal may be allowed to *impersonate* other users and inherit all the privileges granted to them: this implies that, contrary to the RBAC model, the identity of the users is not immaterial and the verification problem becomes more challenging. Moreover, besides regular roles, Liferay also features *site roles*, *organization roles* and *owner roles*, used to constrain access rights exclusively to site members, organization members and resource owners respectively. These special roles have an unconventional semantics, reminiscent of a specific kind of *parametrized roles* [10, 15, 22]. Their introduction breaks a desirable property of the standard RBAC model: user privileges do not depend only on the assigned roles, but also on the state of the Liferay portal, which further complicates verification.

1.2 Contributions

Our contributions can be summarized as follows:

1. we define a formal semantics of Liferay RBAC in terms of a state transition system, which concisely and precisely captures all the subtleties of the access control model. We additionally discuss how we ensure the adequacy of the formal semantics with respect to the behaviour of the real portal;
2. we introduce an abstract semantics which provides a finite approximation of the infinite-state transition system induced by the (concrete) formal semantics. We show that the abstract semantics can be used to soundly verify useful security properties expressed in a fragment of a standard modal (temporal) logic. Moreover, we prove that, when impersonation is not used, the

adoption of the abstract semantics does not introduce any loss of precision in the security analysis;
3. we implement a tool, called LifeRBAC, which leverages the abstract semantics and the modal logic to verify the security of real Liferay portals. We show that the tool is effective at proving the absence of security flaws, while efficient enough to be of practical use on a realistic case study.

Structure of the paper. Section 2 defines the formal semantics of Liferay RBAC. Section 3 introduces the abstract semantics and studies the verification problem. Section 4 presents the tool and the experiments. Section 5 discussed related work. Section 6 concludes. The proofs of the formal results are given in the long version of the paper [4].

2 Semantics of Liferay RBAC

Liferay users are organised into a hierarchy of groups, including companies, organizations and sites. Similarly, different items in the portal are assigned to these groups on an ownership basis: for instance, a given portal page may belong to the site s, which in turn is under the control of the company c. Liferay provides various tools to grant or deny access to a given resource based on the group hierarchy: *scoping* allows to extend access rights on a group to each item belonging to that group, while *parametrized roles* like organization roles and site roles provide facilities for granting access privileges which are restricted to organization-specific/site-specific resources and organization/site members. For example, the assignment of an organization role $\mathsf{Reader}[o]$ may allow members of the organization o to get read access to all the items belonging to o. Finally, owner roles can be used to define access rights for individual resource owners.

2.1 Syntax

We let *Users* be an unbounded set of users and $(G, \preceq)$ be a poset of *groups*. Groups and their underlying order uniformly model different collections handled by the portal, i.e., companies, organizations and sites. We assume an unbounded set *Items*, which includes a number of resources of interest, e.g., portlets, message boards, layouts, etc. Each item i belongs to a fixed set of groups, written $i.groups$.

We assume a set of regular roles *RegRoles* and a set of *role templates* [10]. A role template $r[\cdot] \in RoleTemps$ is a role with a hole (the dot): by instantiating the hole with an object o, we generate a new *parametrized* role $r[o]$. As we formalize below, parametrized roles enforce additional runtime restrictions on the privileges which are granted by the Liferay portal. We assume that role templates are *sorted*, i.e., they are partitioned into two different sets *GrpTemps* and $\{\mathsf{Owner}_j[\cdot]\}_{j \in J}$ (with $J = \{0, \ldots, n\}$ for some natural n) with holes of type G and *Items* respectively: we let *ParRoles* be the set of the parametrized roles obtained by instantiating the holes occurring in role templates with objects of the correct type. We let $R = RegRoles \cup ParRoles$ be the set of roles.

Finally, we let $O = \mathit{Users} \cup \mathit{Items} \cup G \cup R$ be the set of objects. Access to objects is regulated by permissions, drawn from a set Perms. The *scope* of a granted permission can be narrowed or extended using a flag $s \in \{-, \downarrow\}$, which specifies if a permission is given over an individual object or if it can be inherited through the group hierarchy.

Definition 1 (System). *A* system *is a tuple* $\mathcal{S} = (PR, GR, U, I, UR, UG, UU)$*:*

- $PR \subseteq (\mathit{RegRoles} \times \mathit{Perms} \times O \times \{-, \downarrow\}) \cup (\mathit{RoleTemps} \times \mathit{Perms})$ *is the permission-assignment relation;*
- $GR \subseteq G \times \mathit{RegRoles}$ *is a relation mapping groups to regular roles;*
- $U \subseteq \mathit{Users}$ *is a finite set of users;*
- $I \subseteq \mathit{Items}$ *is a finite set of items;*
- $UR \subseteq U \times R$ *is a relation mapping users to their assigned roles;*
- $UG \subseteq U \times G$ *is a relation mapping users to the groups they belong to;*
- $UU \subseteq U \times U$ *is a relation mapping users to users, modelling impersonation.*

By convention we assume that $\forall p \in \mathit{Perms} : (\mathsf{Owner}_0[\cdot], p) \in PR$*, i.e., the first owner role template in the system has full permissions.*

2.2 Semantics

As per previous studies [3, 9], we find it convenient to decouple a system $\mathcal{S}$ into a static *policy* $\mathcal{P}$ and a dynamic *configuration* σ. A policy is a pair $\mathcal{P} = (PR, GR)$, while a configuration is a 5-tuple $\sigma = (U, I, UR, UG, UU)$. The reduction semantics of Liferay RBAC has then the form $\mathcal{P} \vdash \sigma \xrightarrow{\beta} \sigma'$ for some label β.

To specify the semantics, we start by defining to which groups is assigned a given object o under a user-to-group mapping UG. Formally, we inductively define the set $\mathit{groups}_{UG}(o)$ through the self-explanatory inference rules in Table 1.

Table 1 Group Assignment

$$\text{(G-Item)}\ \frac{g \in i.\mathit{groups}}{g \in \mathit{groups}_{UG}(i)} \qquad \text{(G-User)}\ \frac{(u, g) \in UG}{g \in \mathit{groups}_{UG}(u)} \qquad \text{(G-Group)}\ g \in \mathit{groups}_{UG}(g) \qquad \text{(G-Inherit)}\ \frac{g' \preceq g \quad g' \in \mathit{groups}_{UG}(o)}{g \in \mathit{groups}_{UG}(o)}$$

We then define when a user u is granted a permission p over an object o in the system $\mathcal{S}$. The definition of the judgement $\mathcal{S} \vdash \mathit{granted}(u, p, o)$ is in Table 2.

Rule (P-RegI) is standard: it states that, if a user u has a regular role r which grants permission p on the object o, then u has p on o. Rule (P-RegG) allows to extend a permission given over a group g to any object o belonging to g: notice that this must be made explicit in the policy, by using the flag $\downarrow$

Table 2 Permission Granting, where $\mathcal{S} = (PR, GR, U, I, UR, UG, UU)$

$$\text{(P-RegI)}\ \frac{(u,r) \in UR \quad (r,p,o,-) \in PR}{\mathcal{S} \vdash granted(u,p,o)}$$

$$\text{(P-RegG)}\ \frac{(u,r) \in UR \quad (r,p,g,\downarrow) \in PR \quad g \in groups_{UG}(o)}{\mathcal{S} \vdash granted(u,p,o)}$$

$$\text{(P-GroupI)}\ \frac{(u,g) \in UG \quad (g,r) \in GR \quad (r,p,o,-) \in PR}{\mathcal{S} \vdash granted(u,p,o)}$$

$$\text{(P-GroupG)}\ \frac{(u,g) \in UG \quad (g,r) \in GR \quad (r,p,g',\downarrow) \in PR \quad g' \in groups_{UG}(o)}{\mathcal{S} \vdash granted(u,p,o)}$$

$$\text{(P-Template)}\ \frac{(u,r[g]) \in UR \quad (r[\cdot],p) \in PR \quad g \in groups_{UG}(u) \cap groups_{UG}(o)}{\mathcal{S} \vdash granted(u,p,o)}$$

$$\text{(P-Owner)}\ \frac{(u,\mathsf{Owner}_j[i]) \in UR \quad (\mathsf{Owner}_j[\cdot],p) \in PR}{\mathcal{S} \vdash granted(u,p,i)}$$

$$\text{(P-Impersonate)}\ \frac{(u,u') \in UU \quad \mathcal{S}[UU \mapsto \emptyset] \vdash granted(u',p,o)}{\mathcal{S} \vdash granted(u,p,o)}$$

Convention: for any u and i occurring in the judgement we require $u \in U$ and $i \in I$

when assigning the permission. Rules (P-GroupI) and (P-GroupG) are the counterparts of (P-RegI) and (P-RegG) for (regular) roles assigned to groups: they state that any role given to a group is inherited by any user in that group.

Rule (P-Template) is subtle and defines the semantics of parametrized roles: if a role template $r[\cdot]$ is given the permission p and the parametrized role $r[g]$ is assigned to a given user u, then u has p on any object in g, provided that u is himself a member of that group. In this way, a parametrized role $r[g]$ allows to constrain the scope of a permission p inside the group g.

Rule (P-Owner) formalizes the intuition behind owner roles: if a role template $\mathsf{Owner}_j[\cdot]$ is given the permission p and the parametrized role $\mathsf{Owner}_j[i]$ is assigned to a user u for some item i, then u has p on i. Notice that, by the convention in Definition 1, a user with role $\mathsf{Owner}_0[i]$ has full permissions on i.

Finally, rule (P-Impersonate) deals with permissions which are granted upon impersonation: if u is impersonating u' and u' has permission p on the object o, then u has p on o. There is a subtle point to notice though: if u is impersonating u' and u' is impersonating u'', then u is *not* granted the permissions of u''. Formally, this is ensured by emptying the UU component of $\mathcal{S}$ before deriving the judgement in the premises of rule (P-Impersonate). In Liferay, only the permissions which are *statically* known to be granted to u' are inherited by a user impersonating u'.

Having defined when a user is granted a given permission, the formal semantics is relatively simple. The reduction rules are given in Table 3, we just comment the most interesting points. First, owner roles can only be assigned when new items are created and are only removed when items are deleted; we conservatively assume that the owners of dynamically created items have full

Table 3 Reduction Semantics of Liferay RBAC

(Assign-Role)
$$\frac{\mathcal{S} \vdash \mathit{granted}(u_1, \mathsf{AssignRole}, r) \qquad r \neq \mathsf{Owner}_j[i]}{\mathcal{P} \vdash (U, I, UR, UG, UU) \xrightarrow{assign\_role(u_1,u_2,r)} (U, I, UR \cup \{(u_2, r)\}, UG, UU)}$$

(Remove-Role)
$$\frac{\mathcal{S} \vdash \mathit{granted}(u_1, \mathsf{RemoveRole}, r) \qquad r \neq \mathsf{Owner}_j[i]}{\mathcal{P} \vdash (U, I, UR, UG, UU) \xrightarrow{remove\_role(u_1,u_2,r)} (U, I, UR \setminus \{(u_2, r)\}, UG, UU)}$$

(Assign-Group)
$$\frac{\mathcal{S} \vdash \mathit{granted}(u_1, \mathsf{AssignGroup}, g)}{\mathcal{P} \vdash (U, I, UR, UG, UU) \xrightarrow{assign\_group(u_1,u_2,g)} (U, I, UR, UG \cup \{(u_2, g)\}, UU)}$$

(Remove-Group)
$$\frac{\mathcal{S} \vdash \mathit{granted}(u_1, \mathsf{RemoveGroup}, g)}{\mathcal{P} \vdash (U, I, UR, UG, UU) \xrightarrow{remove\_group(u_1,u_2,g)} (U, I, UR, UG \setminus \{(u_2, g)\}, UU)}$$

(Impersonate)
$$\frac{u_1 \notin \mathit{dom}(UU) \qquad \mathcal{S} \vdash \mathit{granted}(u_1, \mathsf{Impersonate}, u_2)}{\mathcal{P} \vdash (U, I, UR, UG, UU) \xrightarrow{impersonate(u_1,u_2)} (U, I, UR, UG, UU \cup \{(u_1, u_2)\})}$$

(Deimpersonate)
$$\mathcal{P} \vdash (U, I, UR, UG, UU) \xrightarrow{deimpersonate(u_1,u_2)} (U, I, UR, UG, UU \setminus \{(u_1, u_2)\})$$

(Add-User)
$$\mathcal{P} \vdash (U, I, UR, UG, UU) \xrightarrow{add\_user(u)} (U \cup \{u\}, I, UR, UG, UU)$$

(Remove-User)
$$\frac{u \notin \mathit{dom}(UU) \qquad UR' = \{(u', r) \in UR \mid u' \neq u\} \qquad UG' = \{(u', g) \in UG \mid u' \neq u\}}{\mathcal{P} \vdash (U, I, UR, UG, UU) \xrightarrow{remove\_user(u)} (U \setminus \{u\}, I, UR', UG', UU)}$$

(Add-Item)
$$\frac{\mathcal{S} \vdash \mathit{granted}(u, \mathsf{AddItem}, g) \qquad g \in i.\mathit{groups}}{\mathcal{P} \vdash (U, I, UR, UG, UU) \xrightarrow{add\_item(u,i)} (U, I \cup \{i\}, UR \cup \{(u, \mathsf{Owner}_0[i])\}, UG, UU)}$$

(Remove-Item)
$$\frac{\mathcal{S} \vdash \mathit{granted}(u, \mathsf{RemoveItem}, i) \qquad UR' = \{(u', r) \in UR \mid r \neq \mathsf{Owner}_j[i]\}}{\mathcal{P} \vdash (U, I, UR, UG, UU) \xrightarrow{remove\_item(u,i)} (U, I \setminus \{i\}, UR', UG, UU)}$$

Notation: we assume $\mathcal{P} = (PR, GR)$ and $\mathcal{S} = (PR, GR, U, I, UR, UG, UU)$

permissions on them, by using the role template $\mathsf{Owner}_0[\cdot]$ in rule (ADD-ITEM). We then notice that each user can only impersonate a single user at a time, by the side-condition $u_1 \notin dom(UU)$ in rule (IMPERSONATE); this implicitly ensures that impersonation is *not* transitive, i.e., if u is impersonating u' and u' can impersonate u'', then u cannot impersonate u''. Finally, when removing a user u, we require that u is not impersonating anyone: this is technically convenient and not limiting, since we can always apply rule (DEIMPERSONATE) up to a configuration where u is impersonating none and then remove him. Notice that no permission is needed to deimpersonate an impersonated user.

We write $\mathcal{P} \vdash \sigma \overset{\overrightarrow{\beta}}{\Longrightarrow} \sigma'$ if and only if there exist $\sigma_1, \ldots, \sigma_{n-1}$ such that $\mathcal{P} \vdash \sigma \xrightarrow{\beta_1} \sigma_1 \wedge \mathcal{P} \vdash \sigma_1 \xrightarrow{\beta_2} \sigma_2 \wedge \ldots \wedge \mathcal{P} \vdash \sigma_{n-1} \xrightarrow{\beta_n} \sigma'$ for some $\overrightarrow{\beta} = \beta_1, \ldots, \beta_n$.

3 Verification of Liferay RBAC

The formal semantics in the previous section can be useful to spot improper privilege escalations by untrusted users of the portal, but it cannot be directly used to prove the *absence* of undesired accesses by an exhaustive state space exploration, since the corresponding labelled transition system has an infinite number of states. We now discuss how we tackle the problem of policy verification by *abstract model-checking* [6].

3.1 A Modal Logic for Verification

We let the syntax of formulas be defined by the following productions:

$$\begin{array}{ll} \textit{State formulas} & \phi ::= \mathit{granted}(u, p, o) \mid \phi \wedge \phi \mid \phi \vee \phi, \\ \textit{Path formulas} & \varphi ::= \Diamond\phi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi. \end{array}$$

This is a simple modal logic, where the modality $\Diamond$ is equivalent to the "finally" operator F available in full-fledged temporal logics like CTL, CTL* or LTL [5]. The (standard) satisfaction relations for state formulas and path formulas are defined by the judgements $\mathcal{P}, \sigma \models \phi$ and $\mathcal{P}, \sigma \models \varphi$ in Table 4. The path formula $\Diamond\phi$ is satisfied by $\mathcal{P}, \sigma$ whenever there exists a reachable configuration from σ under $\mathcal{P}$ where the state formula ϕ holds true.

Though simple, the logic above allows to formalize several standard security properties of interest for RBAC systems. For instance, we have:

- role reachability: user u can never be assigned to regular role r:

$$\mathcal{P}, \sigma \models \neg\Diamond(\mathit{granted}(u, p^*, o^*)),$$

  where p^* is a dummy permission on a dummy object o^* assigned only to r;
- mutual exclusion: user u can never possess both p and p' on object o:

$$\mathcal{P}, \sigma \models \neg\Diamond(\mathit{granted}(u, p, o) \wedge \mathit{granted}(u, p', o))$$

Table 4 Satisfaction Relation

$$\text{(LS-Basic)}\ \frac{\mathcal{P},\sigma \vdash \mathit{granted}(u,p,o)}{\mathcal{P},\sigma \models \mathit{granted}(u,p,o)} \qquad \text{(LS-And)}\ \frac{\mathcal{P},\sigma \models \phi_1 \quad \mathcal{P},\sigma \models \phi_2}{\mathcal{P},\sigma \models \phi_1 \wedge \phi_2} \qquad \text{(LS-Or)}\ \frac{\mathcal{P},\sigma \models \phi_i}{\mathcal{P},\sigma \models \phi_1 \vee \phi_2}$$

$$\text{(LP-Finally)}\ \frac{\mathcal{P} \vdash \sigma \overset{\vec{\beta}}{\Longrightarrow} \sigma' \qquad \mathcal{P},\sigma' \models \phi}{\mathcal{P},\sigma \models \Diamond\phi} \qquad \text{(LP-Not)}\ \frac{\mathcal{P},\sigma \not\models \varphi}{\mathcal{P},\sigma \models \neg\varphi} \qquad \text{(LP-And)}\ \frac{\mathcal{P},\sigma \models \varphi_1 \quad \mathcal{P},\sigma \models \varphi_2}{\mathcal{P},\sigma \models \varphi_1 \wedge \varphi_2} \qquad \text{(LP-Or)}\ \frac{\mathcal{P},\sigma \models \varphi_i}{\mathcal{P},\sigma \models \varphi_1 \vee \varphi_2}$$

Notation: in rules (LS-Or) and (LP-Or) we let $i \in \{1,2\}$

– group reachability: user u can never join group g:

$$\mathcal{P},\sigma \models \neg\Diamond(\mathit{granted}(u,p^*,g)),$$

where p^* is a dummy permission assigned only to the dummy role template $r[\cdot]$ and the parametrized role $r[g]$ is assigned to u in the configuration σ.

3.2 Abstract Semantics

The abstract semantics builds on two core ideas. First, we observe that the actual identity of users and items is often immaterial: for instance, two items belongings to the same groups behave exactly in the same way for Liferay RBAC. Second, many transitions of the semantics (e.g., removing roles) actually weaken the privileges granted to a given user, hence they are irrelevant to detect security violations. Leveraging these two observations, the abstract semantics consists of: (i) a finite-range abstraction function $\alpha : O \to O$, mapping each object in the unbounded set O to some canonical representative; and (ii) an abstract reduction relation, defining the dynamics of configurations abstracted by α. The abstraction function can be chosen arbitrarily, as long as it satisfies some syntactic conditions given below: one may choose different trade-offs between precision and efficiency by using different abstractions for verification.

We presuppose two functions $\alpha_u : \mathit{Users} \to \mathit{Users}$ and $\alpha_i : \mathit{Items} \to \mathit{Items}$. We then build on top of them an abstraction function $\alpha : O \to O$ as follows:

$$\alpha(o) = \begin{cases} \alpha_u(u) & \text{if } o \text{ is a user } u \\ \alpha_i(i) & \text{if } o \text{ is an item } i \\ r[\alpha_i(i)] & \text{if } o \text{ is a parametrized role } r[i] \\ o & \text{otherwise.} \end{cases}$$

We extend α to formulas, (sequences of) labels, tuples and sets by applying it to any object syntactically occurring therein.

The *abstract* reduction relation $\mathcal{P} \vdash \sigma \overset{\alpha}{\mapsto} \sigma'$ is obtained from the rules in Table 3 by excluding (Remove-Role), (Remove-Group), (Remove-User) and (Remove-Item), and by dropping the side-condition $u_1 \notin dom(UU)$ from rule (Impersonate).

We let $\mathcal{P} \vdash \sigma \overset{\overrightarrow{\beta}}{\Longmapsto} \sigma'$ be the obvious generalization to the abstract semantics of the relation $\mathcal{P} \vdash \sigma \overset{\overrightarrow{\beta}}{\Rightarrow} \sigma'$ defined above. We then let $\mathcal{P}, \sigma \models_\alpha \varphi$ be the satisfaction relation obtained from the rules in Table 4 by replacing (LP-Finally) with:

$$\text{(ALP-Finally)} \quad \frac{\alpha(\mathcal{P}) \vdash \alpha(\sigma) \overset{\alpha(\overrightarrow{\beta})}{\Longmapsto} \sigma' \qquad \alpha(\mathcal{P}), \sigma' \models \alpha(\phi)}{\mathcal{P}, \sigma \models_\alpha \Diamond\phi}$$

and by introducing the obvious counterparts of rules (LP-Not), (LP-And) and (LP-Or). Since any abstraction function has a finite range, it is easy to prove:

Lemma 1. *There exists a decision procedure* Abs-Sat$(\alpha, \mathcal{P}, \sigma, \varphi)$ *for* $\mathcal{P}, \sigma \models_\alpha \varphi$.

We verify security properties of the infinite-state concrete semantics by model-checking the finite-state abstract semantics. To isolate the fragment of the modal logic amenable for verification, we let negation-free formulas $\widehat{\varphi}$ and rank-1 formulas ψ be defined by the following productions:

$$\begin{array}{ll} \textit{Negation-free formulas} & \widehat{\varphi} ::= \Diamond\phi \mid \widehat{\varphi} \wedge \widehat{\varphi} \mid \widehat{\varphi} \vee \widehat{\varphi}, \\ \textit{Rank-1 formulas} & \psi ::= \neg\widehat{\varphi} \mid \psi \wedge \psi \mid \psi \vee \psi. \end{array}$$

We can construct a procedure which determines if an arbitrary rank-1 formula is satisfied or not by the concrete semantics (see Fig. 1). The next subsections discuss sufficient conditions for the soundness and the completeness of the algorithm, i.e., conditions on the policy $\mathcal{P}$ and the abstraction function α which ensure that a positive (resp. negative) answer by Sat$(\alpha, \mathcal{P}, \sigma, \psi)$ implies that ψ is satisfied (resp. not satisfied) by $\mathcal{P}, \sigma$. Notice that all the example properties previously described are expressed by a rank-1 formula.

```
Sat(α, P, σ, ψ):
    match ψ with
    | ¬φ̂        →  not Abs-Sat(α, P, σ, φ̂)
    | ψ1 ∧ ψ2   →  Sat(α, P, σ, ψ1) and Sat(α, P, σ, ψ2)
    | ψ1 ∨ ψ2   →  Sat(α, P, σ, ψ1) or Sat(α, P, σ, ψ2)
```

Fig. 1. Abstract Model-Checking Algorithm

3.3 Soundness of Verification

We first prove the *soundness* of the algorithm in Fig. 1, i.e., we show that, assuming a mild syntactic restriction on the abstraction function α, a positive answer by $\textsc{Sat}(\alpha, \mathcal{P}, \sigma, \psi)$ implies that $\mathcal{P}, \sigma \models \psi$ holds true.

Definition 2 (Group-preserving Abstraction). *An abstraction function α is* group-preserving *iff $\forall i \in Items : i.groups \subseteq \alpha(i).groups$.*

Definition 3 (Permission-based Ordering). *We let $\sigma \sqsubseteq_{\mathcal{P}} \sigma'$ if and only if $\mathcal{P}, \sigma \vdash granted(u, p, o)$ implies $\mathcal{P}, \sigma' \vdash granted(u, p, o)$ for any u, p and o.*

The next theorem states that any behaviour of the concrete semantics has a counterpart in the abstract semantics. It also ensures that the abstract semantics over-approximates the permissions granted to each user. The result would not hold in general if the side-condition of rule (Impersonate) was included in the abstract semantics: we omit further technical details due to space constraints.

Theorem 1 (Soundness). *Let α be group-preserving. If $\mathcal{P} \vdash \sigma \overset{\overrightarrow{\beta}}{\Longrightarrow} \sigma'$, then there exists a sub-trace of $\overrightarrow{\beta}$, call it $\overrightarrow{\gamma}$, such that $\alpha(\mathcal{P}) \vdash \alpha(\sigma) \overset{\alpha(\overrightarrow{\gamma})}{\Longrightarrow} \sigma''$ for some σ'' such that $\alpha(\sigma') \sqsubseteq_{\alpha(\mathcal{P})} \sigma''$.*

Using the theorem above, we can prove the soundness of verification. Notice that the only assumption needed for soundness is on the abstraction function α: the result applies to any choice of $\mathcal{P}$, σ and ψ.

Theorem 2 (Sound Verification). *Let α be a group-preserving abstraction function. If $\textsc{Sat}(\alpha, \mathcal{P}, \sigma, \psi)$ returns a positive answer, then $\mathcal{P}, \sigma \models \psi$.*

3.4 Completeness of Verification

We now identify conditions for the *completeness* of the algorithm in Fig. 1, i.e., we discuss under which assumptions a negative answer by $\textsc{Sat}(\alpha, \mathcal{P}, \sigma, \psi)$ ensures that $\mathcal{P}, \sigma \not\models \psi$. This is important for the *precision* of the analysis.

To state and prove the completeness result, we focus on a particular class of policies which does not allow to impersonate users. We remark that this corresponds to a realistic use case, since Liferay can be configured to prevent impersonation by setting the property `portal.impersonation.enable` to false in the file `webapps/ROOT/WEB-INF/classes/portal-developer.properties`.

Definition 4 (Impersonation-free Policy/Configuration). *A policy $\mathcal{P} = (PR, GR)$ is* impersonation-free *iff $\forall (r, p, o, s) \in PR : p \neq \mathsf{Impersonate}$ and $\forall (r[\cdot], p) \in PR : p \neq \mathsf{Impersonate}$. A configuration $\sigma = (U, I, UR, UG, UU)$ is* impersonation-free *iff $UU = \emptyset$.*

Given a configuration $\sigma = (U, I, UR, UG, UU)$, let: $users(\sigma) = U$; $items(\sigma) = I$; $groups_\sigma(u) = \{g \mid (u, g) \in UG\}$; and $roles_\sigma(u) = \{r \mid (u, r) \in UR\}$.

The completeness result requires to find sufficient conditions which ensure that the abstract semantics is under-approximating the concrete semantics. If impersonation is never used, it is enough to require that each user in the abstract semantics belongs to fewer groups and has fewer roles than one of his corresponding users in the concrete semantics, as formalized next.

Definition 5 (Abstract Under-Approximation). *A configuration σ is an* abstract under-approximation *of a configuration σ', written $\sigma \lesssim_\alpha \sigma'$, if and only if both the following conditions hold true:*

- $\forall u \in users(\sigma) : \exists u' \in users(\sigma') : u = \alpha(u') \wedge groups_\sigma(u) \subseteq groups_{\sigma'}(u') \wedge roles_\sigma(u) \subseteq \alpha(roles_{\sigma'}(u'))$;
- $\forall i \in items(\sigma) : \exists i' \in items(\sigma') : i = \alpha(i')$.

Definition 6 (Group-forgetting Abstraction). *An abstraction function α is* group-forgetting *iff $\forall i \in Items : \alpha(i).groups \subseteq i.groups$.*

The next theorem states that any behaviour in the abstract semantics has a counterpart in the concrete semantics, assuming that impersonation is never used. It additionally ensures that the desired under-approximation is preserved upon reduction.

Theorem 3 (Completeness). *Let $\mathcal{P}, \sigma$ be impersonation-free and let α be group-forgetting. If $\alpha(\sigma) \lesssim_\alpha \sigma$ and $\alpha(\mathcal{P}) \vdash \alpha(\sigma) \xRightarrow{\alpha(\overrightarrow{\beta})} \sigma'$ for some $\overrightarrow{\beta}$ and σ', then there exists σ'' such that $\mathcal{P} \vdash \sigma \xRightarrow{\overrightarrow{\beta}} \sigma''$ and $\sigma' \lesssim_\alpha \sigma''$.*

We need also an additional condition to prove the completeness of verification: we must ensure that the identity of any object occurring in the formula ψ to verify is *respected* by the abstraction function, in the following sense.

Definition 7 (Respectful Abstraction). *An abstraction function α* respects *an object o iff $\alpha(o) = o$ and $\forall o' \in O : o' \neq o \Rightarrow \alpha(o') \neq o$. An abstraction function α* respects *ψ iff it respects any object occurring in ψ.*

Theorem 4 (Complete Verification). *Let $\mathcal{P}, \sigma$ be impersonation-free and let α be a group-forgetting abstraction function which respects ψ. If $\alpha(\sigma) \lesssim_\alpha \sigma$ and $\mathcal{P}, \sigma \models \psi$, then $\textsc{Sat}(\alpha, \mathcal{P}, \sigma, \psi)$ returns a positive answer.*

Completeness of verification does not hold in general if impersonation is used. Specifically, if a user u is allowed to impersonate both u_1 and u_2, he will be able to do it at the same time in the abstract semantics, thus getting the union of their privileges; however, the two users cannot be impersonated at the same time in the concrete semantics, which breaks the intended under-approximation.

4 Implementation: LifeRBAC

LifeRBAC is a Liferay plugin providing a simple user interface to let portal administrators input security queries about the underlying RBAC system. The plugin takes a snapshot of the portal and translates it into a corresponding representation in our formal model, encoded in the ASLAN specification language [18]. The initial state representation is joined with a set of (hand-coded) transition rules, corresponding to the ASLAN implementation of the abstract semantics, and the query is translated into a modal logic formula, which is verified using the state-of-art model-checker SATMC [1].

4.1 Implementation Details

LifeRBAC currently supports two different analyses, corresponding to the choice of two different abstraction functions. In the *fast* analysis, only the identity of the users occurring in the security query is preserved by the abstraction function, while all the other users are abstracted into a super-user with the union of their privileges. In the *precise* analysis, the identity of the users occurring in the security query is still preserved, but all the other users are abstracted into a canonical representative sharing their same groups and roles. As to items, in both cases we preserve their identity when they occur in the security query, while we abstract them into a canonical representative sharing the same groups otherwise. Observe that both the choices of the abstraction function satisfy the conditions of Theorem 2, hence both the analyses are *sound*. Moreover, the precise analysis satisfies also the conditions of Theorem 4, hence it is *complete* for any policy which does not allow to impersonate users.

At the moment LifeRBAC only supports the verification of security queries predicating over a subset of the objects available in Liferay, i.e., users, groups and layouts. Including additional types of objects (e.g., portlets) is essentially a matter of programming.

4.2 Experiments

Inspired by a previously published case study [23], we consider an experimental setting modelling a hypothetical university with 3 departments and 10 courses. We represent the university as the only company in the portal and the departments as three different organizations; then, we create a private site for each department and a corresponding child site for every course. We consider 15 role templates: 6 templates are used to generate site roles, while 9 templates are used to generate organization roles. We also include 14 regular roles to collect permissions which are not scoped to any specific site or organization. Overall, we have $6 \cdot 13 + 9 \cdot 3 + 14 = 119$ roles; for each of them, we model access permissions to different resources as read/write privileges on specific web pages in the sites, and we enable administrative permissions where appropriate, e.g., a user with role $\mathsf{Professor}[c]$ for some course c can assign the parametrized role $\mathsf{Student}[c]$. Finally, we create 1000 users with different role combinations.

In the experiments, we consider three different security queries:

- q_1: can student u_1, who is a member of site s_1, delete a page from s_1?
- q_2: can student u_2 join site s_2 belonging to another department?
- q_3: can student u_3, who is a member of site s_3, assign a user u_4 to s_3?

All the three queries are performed against three slightly different configurations of increasing complexity. In the first configuration no user is allowed to impersonate other users or to assign members to existing sites/organizations. In the second configuration clerks can move users along the group hierarchy, but no impersonation is possible, while in the third configuration administrators are allowed to impersonate any user. The last scenario is a "stress test" for our tool, since unconstrained impersonation leads to a state-space explosion.

The time required to check the three queries against the three described configurations and their results are given in Table 5. In the table, we also keep track of the number of users obtained after applying the abstraction; all the attacks have been confirmed by hand on Liferay 6.2 CE. The experiments have been performed on an Intel Xeon 2.4Ghz running Ubuntu Linux 12.04 LTS.

Table 5 Experimental Results

		Fast Analysis				Precise Analysis			
conf.	query	#users	time	attack	real	#users	time	attack	real
A	q_1	3	1m 58s	yes	yes	62	6m 10s	yes	yes
	q_2	3	1m 46s	no	-	62	1m 52s	no	-
	q_3	3	1m 48s	no	-	62	1m 53s	no	-
B	q_1	3	2m 40s	yes	yes	62	49m 01s	yes	yes
	q_2	3	1m 50s	yes	yes	62	21m 46s	yes	yes
	q_3	3	1m 50s	no	-	62	22m 30s	no	-
C	q_1	3	2m 53s	yes	yes	62	182m 21s	yes	yes
	q_2	3	1m 59s	yes	yes	62	56m 02s	yes	yes
	q_3	3	2m 37s	no	-	62	54m 46s	no	-

A = no impersonate and no groups; B = only groups; C = groups and impersonate
#users = the number of users after the abstraction; real = attack confirmed

For the first configuration, where we do not include Liferay-specific features, the verification time is very good and in line with previous work on standard RBAC systems [9]. In the worst case, verification takes around 6 minutes and we are able to *prove* the existence of an attack by the completeness result, something which is beyond previous abstraction-based proposals. Based on the numbers in the table, we observe that the possibility of assigning and removing groups, thus activating and deactivating parametrized roles, has a significant impact on the performances, especially for the precise analysis. Impersonation further contributes to complicate the verification problem, as the results for the precise analysis clearly highlight, but it does not hinder too much the performances of

the fast analysis. Remarkably, despite the huge approximation applied by the fast analysis, we did not identify false positives for these queries (and also for other tests we performed). We leave as future work further study on the precise analysis, to improve its performance by the usage of static slicing techniques [8].

4.3 Discussion: Adequacy of the Semantics

A thorny issue when verifying a complex framework like Liferay RBAC is bridging the gap between the formal model and the real system. In particular, observe that: (*i*) Liferay features many different permissions and it is not obvious which ones correspond to the few administrative permissions (e.g., AssignRole) we include in the model; and (*ii*) the Liferay permission checker is a complicated piece of code, which selectively grants or denies some permissions based on the *type* of the object of an access request, but types have only a marginal role in the permission granting process formalized in Table 2.

The key insight is that both the problems above can be dealt with just by carefully constructing the permission-assignment relation PR used in the formal model. We assessed the adequacy of our solution by *testing*, an idea proposed by the programming languages community [12]. Specifically, we created a tool, LR-Test, which takes a snapshot of the Liferay portal and constructs its corresponding representation in our formal model. The tool systematically queries both the Liferay permission checker and its formal counterpart (deriving the judgements in Table 2) to detect any mismatch on resource accesses and enabled administrative actions. Mismatches may be of two types: false positives lead to an over-conservative security analysis, while false negatives lead to overlooking real security flaws. In the current prototype, we eliminated all the false negatives we identified and we only left a few false positives to be fixed.

5 Related Work

Abstraction techniques as an effective tool for RBAC verification have been first concurrently proposed by Bugliesi *et al.* [3] and Ferrara *et al.* [9]. The first paper proposes a formal semantics for grsecurity, an RBAC system built on top of the Linux kernel, and then uses abstract model-checking to verify some specific security properties of interest. Notably, the abstraction adopted in [3] is fixed, while the results in this paper are parametric with respect to the choice of an abstraction function. The authors of [9], instead, do not apply model-checking techniques, but rather construct an imperative program abstracting the dynamics of the RBAC system and apply standard results from program verification to soundly approximate the role reachability problem. The proposed abstraction is incomplete, i.e., the analysis may produce false positives. In recent work [8], the same research group presented VAC, a tool for role reachability analysis, which can be used both to prove RBAC policies correct and to find errors in them (adopting different analysis backends).

A different approach to RBAC verification is based on SMT solving. Armando and Ranise proposed a symbolic verification framework for RBAC systems with an unbounded number of users [2]. More recent research holds great promise in making similar techniques scale to verify large RBAC policies [19].

Error finding in RBAC policies is complementary to verification and abstraction techniques have proved fundamental also for this problem, since plain model-checking does not scale to the analysis of real systems. The most known proposal in the area is MOHAWK, a tool based on an abstraction-refinement model-checking strategy [13]. Interestingly, MOHAWK has been recently extended to prove RBAC policies correct [14]. The analysis is limited to separate administration policies, a restriction which we do not assume in this paper.

To overcome the performance issues affecting RBAC policy verification, many authors identified tractable fragments of the general RBAC model and presented different algorithms to answer useful security queries under specific policy restrictions [23, 11, 21, 16]. Most of these results are proved for a finite set of users, while our work assumes an unbounded set of users entering and leaving the system.

6 Conclusion

We presented a formal semantics for Liferay RBAC and we tackled the verification problem through abstract model-checking, discussing sufficient conditions for the soundness and the completeness of the analysis. We then implemented a tool, LIFERBAC, which can be used to verify the security of real Liferay portals and we reported on experiments showing the effectiveness of our solution.

As a future work, we plan to strengthen the completeness result to policies involving impersonation. Moreover, we want to extend LIFERBAC to support error finding in the underlying RBAC policy, to include a counter-example generation module for flawed policies, and to make it significantly faster by adapting static slicing techniques from the literature [8].

Acknowledgements. This research is partially supported by the MIUR projects ADAPT and CINA. Alvise Rabitti acknowledges support by SMC Treviso. We would like to thank Rocco Germinario and Paolo Valeri of SMC Treviso for participating to many technical discussions about Liferay.

References

1. Armando, A., Carbone, R., Compagna, L.: SATMC: A SAT-based model checker for security-critical systems. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 31–45 (2014)
2. Armando, A., Ranise, S.: Automated symbolic analysis of ARBAC-policies. In: Security and Trust Management (STM). pp. 17–34 (2010)
3. Bugliesi, M., Calzavara, S., Focardi, R., Squarcina, M.: Gran: Model checking grsecurity RBAC policies. In: Computer Security Foundations (CSF). pp. 126–138 (2012)

4. Calzavara, S., Rabitti, A., Bugliesi, M.: Formal verification of Liferay RBAC (full version). Available at www.dais.unive.it/~calzavara/papers/essos15-full.pdf
5. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. 8(2), 244–263 (1986)
6. Cousot, P., Cousot, R.: Refining model checking by abstract interpretation. Autom. Softw. Eng. 6(1), 69–95 (1999)
7. Ferraiolo, D.F., Sandhu, R.S., Gavrila, S.I., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. 4(3), 224–274 (2001)
8. Ferrara, A.L., Madhusudan, P., Nguyen, T.L., Parlato, G.: Vac - verifier of administrative role-based access control policies. In: Computer Aided Verification (CAV). pp. 184–191 (2014)
9. Ferrara, A.L., Madhusudan, P., Parlato, G.: Security analysis of role-based access control through program verification. In: Computer Security Foundations (CSF). pp. 113–125 (2012)
10. Giuri, L., Iglio, P.: Role templates for content-based access control. In: ACM Workshop on Role-Based Access Control. pp. 153–159 (1997)
11. Gofman, M.I., Luo, R., Solomon, A.C., Zhang, Y., Yang, P., Stoller, S.D.: Rbacpat: A policy analysis tool for role based access control. In: Tools and Algorithms for the Construction and the Analysis of Systems (TACAS). pp. 46–49 (2009)
12. Guha, A., Saftoiu, C., Krishnamurthi, S.: The essence of JavaScript. In: European Conference on Object Oriented Programming (ECOOP). pp. 126–150 (2010)
13. Jayaraman, K., Ganesh, V., Tripunitara, M.V., Rinard, M.C., Chapin, S.J.: Automatic error finding in access-control policies. In: ACM Conference on Computer and Communications Security (CCS). pp. 163–174 (2011)
14. Jayaraman, K., Tripunitara, M.V., Ganesh, V., Rinard, M.C., Chapin, S.J.: Mohawk: Abstraction-refinement and bound-estimation for verifying access control policies. ACM Trans. Inf. Syst. Secur. 15(4), 18 (2013)
15. Li, N., Mitchell, J.C.: A role-based trust-management framework. In: DARPA Information Survivability Conference and Exposition (DISCEX). pp. 201–212 (2003)
16. Li, N., Tripunitara, M.V.: Security analysis in role-based access control. ACM Trans. Inf. Syst. Secur. 9(4), 391–420 (2006)
17. Liferay Inc.: Liferay clients and case studies, available online at `https://www.liferay.com/it/products/liferay-portal/stories`
18. Mödersheim, S.: Deciding security for a fragment of ASLan. In: European Symposium on Research in Computer Security (ESORICS). pp. 127–144 (2012)
19. Ranise, S., Truong, A.T., Armando, A.: Boosting model checking to analyse large ARBAC policies. In: Security and Trust Management (STM). pp. 273–288 (2012)
20. Sandhu, R.S., Bhamidipati, V., Munawer, Q.: The ARBAC97 model for role-based administration of roles. ACM Trans. Inf. Syst. Secur. 2(1), 105–135 (1999)
21. Sasturkar, A., Yang, P., Stoller, S.D., Ramakrishnan, C.R.: Policy analysis for administrative role-based access control. Theor. Comput. Sci. 412(44), 6208–6234 (2011)
22. Stoller, S.D., Yang, P., Gofman, M.I., Ramakrishnan, C.R.: Symbolic reachability analysis for parameterized administrative role-based access control. Computers & Security 30(2-3), 148–164 (2011)
23. Stoller, S.D., Yang, P., Ramakrishnan, C.R., Gofman, M.I.: Efficient policy analysis for administrative role based access control. In: ACM Conference on Computer and Communications Security (CCS). pp. 445–455 (2007)