Static Detection of Collusion Attacks in ARBAC-based Workflow Systems

Stefano Calzavara Alvise Rabitti Enrico Steffinlongo Michele Bugliesi
Università Ca' Foscari Venezia

***Abstract*—Authorization in workflow systems is usually built on top of role-based access control (RBAC); security policies on workflows are then expressed as constraints on the users performing a set of tasks and the roles assigned to them. Unfortunately, when role administration is distributed and potentially untrusted users contribute to the role assignment process, like in the case of Administrative RBAC (ARBAC), collusions may take place to circumvent the intended workflow security policies. In a collusion attack, a set of users of a workflow system collaborates by changing the user-to-role assignment, so as to sidestep the security policies and run up to completion a workflow they could not complete otherwise.**

In this paper, we study the problem of collusion attacks in a formal model of workflows based on stable event structures and we define a precise notion of security against collusion. We then propose a static analysis technique based on a reduction to a role reachability problem for ARBAC, which can be used to prove or disprove security for a large class of workflow systems. We also discuss how to aggressively optimise the obtained role reachability problem to ensure its tractability. Finally, we implement our analysis in a tool, WARBAC, and we experimentally show its effectiveness on a set of publicly available examples, including a realistic case study.

I. Introduction

A workflow is a temporally organised collection of tasks, representing a business process specification. Workflow systems are software supporting a specific set of business processes through the execution of computerized task definitions. These software not only ensure that the execution of the tasks in a workflow respects the expected temporal order, but also that these tasks are performed by *authorized* users.

Authorization in workflow systems is usually built on top of role-based access control (RBAC). RBAC is a very natural choice for workflow systems, since roles provide a convenient abstraction to represent a (possibly large) set of users entitled to perform a given task [4], [7]. When role-based security policies on task execution are not expressive enough, security policies on workflows can also include constraints on the identity of the users performing a set of tasks, like binding-of-duty and separation-of-duty constraints [8], [19]. Binding-of-duty (BoD) constraints enforce two different tasks to be performed by the same user, e.g., to prevent an undesired disclosure of sensitive information or to ensure a single user takes full responsibility for a set of related tasks. Separation-of-duty (SoD) constraints, instead, play the dual role of ensuring that two different tasks are performed by two different users, e.g., to prevent frauds or conflicts of interest.

An important observation for security is that, though both role-based and identity-based security policies like BoD and SoD constraints are static and declarative in nature, the set of roles assigned to the users of a workflow system is typically not. For instance, the Administrative RBAC (ARBAC) standard [12] allows system administrators to specify which roles are entitled to assign other roles to users, based on the sets of roles assigned (or not assigned) to them; similarly, roles may be granted the ability of revoking other roles from system users. Role administration is thus highly distributed in ARBAC systems, which is a very desirable feature for normal system functionality; however, it is also well-known that such a feature poses an important security challenge, since the sets of roles which may be dynamically assigned to users is very hard to predict without automated tool support [17], [20].

In the case of workflow systems based on the ARBAC model, the fact that potentially untrusted users contribute to the role assignment process enables hard-to-spot *collusions* aimed at circumventing the intended workflow security policies. Specifically, in a collusion attack a set of users of a workflow system collaborates by changing the user-to-role assignment, so as to sidestep the security policies put in place by the system administrators and run up to completion a workflow they could not complete otherwise.

A. Motivating Example

We graphically represent workflows as directed graphs, including one node per task, a start node $\cdot$ and an end node $\checkmark$. We use directed arrows to represent temporal dependencies, dashed lines labelled with $\#$ to visualize exclusive choices and dashed lines labelled with either $=$ and $\neq$ to represent BoD and SoD constraints respectively. We annotate each task with a subscript including the set of roles entitled to perform it. For instance, Figure 1 represents a workflow with three tasks a, b, c, which can be performed by any user who is granted role $\mathsf{R}_1, \mathsf{R}_2, \mathsf{R}_3$ respectively. After the execution of task a, the workflow offers an exclusive choice between tasks b and c and, no matter which task is chosen, the second task must be performed by the same user who performed a.

Consider now two users u_1 and u_2 who are assigned roles $\mathsf{R}_1, \mathsf{R}_2$ respectively. These users cannot complete the workflow just with their roles, since they lack the privileges needed to perform a, b or a, c without violating the BoD constraints. However, assume that any user who is assigned role R_2 is also allowed to assign role R_3 to any user owning role R_1: this kind of policies is common in access control systems supporting role administration, including the standard ARBAC model [12]. Under this policy, users u_1 and u_2 can collude to

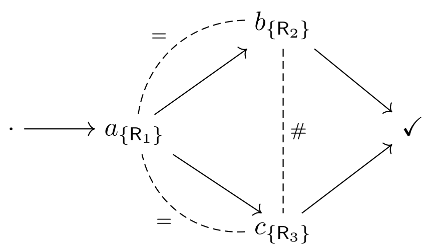

Figure 1. Example of workflow

complete the workflow as follows: user u_2 assigns role R_3 to user u_1, who thus becomes able to perform tasks a and c on her own. Though such a possibility may be easy to spot in this simple example, detecting collusions is hard in general, since they may be enabled by arbitrarily long sequences of actions and the underlying ARBAC policy may include hundreds of rules for role administration like the one we discussed.

B. Contributions

In this paper, we make the following contributions:

1) we propose a formal model of workflows based on *stable event structures* [24], which generalizes previous proposals based on partial orders [21], [22]. We then integrate this model with ARBAC, by defining a small-step operational semantics for workflow systems where workflow actions (execution of a task) and administrative actions (assignment or revocation of roles) are arbitrarily interleaved;
2) we define a precise notion of security against collusion for workflow systems, ensuring that administrative actions cannot be abused to sidestep the workflow security policy. The definition is adapted from previous work on the security of delegation in access control systems [23];
3) we propose a static analysis based on a reduction to a role reachability problem for ARBAC, which can be used to prove or disprove security against collusion for restricted yet useful classes of workflow systems. By reducing security to role reachability, it is possible to reuse available tools for role reachability analysis [13], [9], [16] to effectively check it. We aggressively optimise the role reachability problem to ensure its tractability;
4) we implement our static analysis in a tool, WARBAC, and we experimentally show its effectiveness on a set of publicly available examples, including a realistic case study describing a first-aid procedure.

We make WARBAC, all the experimental data and an extended version of the present paper (with proofs) available online [10].

Structure of the Paper: Section II presents the operational model. Section III introduces the formal definition of security and gives an example. Section IV details the reduction to role reachability and its optimization. Section V shows a few example reductions. Section VI presents WARBAC and reports on the experimental results. Section VII discusses related work. Section VIII concludes and hints at future work.

II. Operational Model

Our operational model is obtained by integrating a standard ARBAC model, as formalized, e.g., in [9], [13], with a workflow represented as a *stable event structure* [24], extended with a security policy assigning required roles to tasks and supporting both BoD and SoD constrains.

A. ARBAC

We presuppose the existence of finite sets of users U and roles R.

Definition 1 (ARBAC Policy). *An* ARBAC policy *is a pair $\mathcal{P} = \langle CA, CR\rangle$, where $CA \subseteq R \times 2^R \times 2^R \times R$ is a can-assign relation and $CR \subseteq R \times R$ is a can-revoke relation.*

A can-assign rule $(r_a, R_p, R_n, r_t) \in CA$ states that a user with role r_a can assign role r_t to any user who has all the roles in the set R_p (the *positive* preconditions) and none of the roles in the set R_n (the *negative* preconditions). A can-revoke rule $(r_a, r_t) \in CR$, instead, states that a user with role r_a can unconditionally revoke role r_t from any user.

Definition 2 (ARBAC System). *An* ARBAC system *is a pair $\mathcal{S} = \langle \mathcal{P}, UR\rangle$, where $\mathcal{P}$ is an ARBAC policy and $UR \subseteq U \times R$ is an initial user-to-role assignment.*

For any user u, let $UR(u) = \{r \,|\, (u, r) \in UR\}$. The operational semantics of an ARBAC system $\mathcal{S} = \langle \mathcal{P}, UR\rangle$ is defined by the changes which can be performed to the initial user-to-role assignment UR according to the policy $\mathcal{P}$. This is defined by the reduction relation $\mathcal{P} \rhd UR \rightsquigarrow UR'$ in Table I, providing the formal counterpart of the intuitions above.

Table I Reduction semantics for ARBAC ($\mathcal{P} \rhd UR \rightsquigarrow UR'$)

(R-Assign)
$$\frac{(u_a, r_a) \in UR \qquad (r_a, R_p, R_n, r_t) \in CA \qquad R_p \subseteq UR(u) \qquad R_n \cap UR(u) = \emptyset}{\langle CA, CR\rangle \rhd UR \rightsquigarrow UR \cup \{(u, r_t)\}}$$

(R-Revoke)
$$\frac{(u_a, r_a) \in UR \qquad (r_a, r_t) \in CR}{\langle CA, CR\rangle \rhd UR \rightsquigarrow UR \setminus \{(u, r_t)\}}$$

B. Workflows

We propose a general, abstract model of workflows based on *stable event structures*, a true concurrency model originally proposed by Winskel [24]. Stable event structures are very appealing candidates to represent workflows, since they can naturally model sequential and parallel execution of tasks, as well as non-deterministic choices [18], [24]. Moreover, stable event structures are more expressive than previous models of workflows based on a partially ordered set of tasks [21], [22], since the latter correspond to *elementary event structures* [25] and hence cannot represent non-determinism, which instead is a desirable feature for many workflows (including the one in

Figure 1). Besides being an expressive model on their own, stable event structures have also been proposed to define the semantics of several formalisms commonly used to model workflows, including CCS, CSP and Petri nets (see [24] for CCS/CSP and [5] for Petri nets). By working with stable event structures, we make our theory general enough to be directly applicable to any workflow specification language whose semantics can be defined in terms of these structures.

There are several slightly different definitions of stable event structure in the literature, the one we use here is taken from [18]: it is more compact than the original formulation in [24], but semantically equivalent to it. We just omit the labelling function from the definition, since it does not play any role in the present paper.

Definition 3 (Stable Event Structure)**.** *A* stable event structure *is a triple $\mathcal{E} = \langle E, \#, \vdash \rangle$, where:*

1) *E is a denumerable set of events;*
2) *$\# \subseteq E \times E$ is a symmetric, irreflexive* conflict *relation;*
3) *$\vdash \subseteq 2^E \times E$ is an* enabling *relation;*
4) *for all events $e \in E$ and all sets of events $X, Y \subseteq E$ such that $X \neq Y$, the following* stability axiom *is satisfied:*

$$X \vdash e \land Y \vdash e \Rightarrow \exists\, e_1, e_2 \in X \cup Y : (e_1, e_2) \in \#.$$

If $X \vdash e$ for some set of events $X \subseteq E$, then the occurrence of all the events in X enables the occurrence of the event e; if $(e_1, e_2) \in \#$, instead, then the occurrence of event e_1 rules out the occurrence of event e_2 and vice-versa. The stability axiom requires that, if there are different enablings for the same event, they are conflicting, which ensures that each event is enabled in an essentially unique way.

The semantics of stable event structures is defined in terms of a set of *configurations*, defined as follows [24].

Definition 4 (Configuration)**.** *Given a stable event structure $\mathcal{E} = \langle E, \#, \vdash \rangle$, a* configuration *of $\mathcal{E}$ is a finite set of events $X \subseteq E$ such that:*

1) *$\forall\, e, e' \in X : (e, e') \notin \#$;*
2) *$\forall\, e \in X. \exists\, e_1, \ldots, e_n \in X : e_n = e \land \forall\, i \leq n. \exists\, Y \subseteq \{e_1, \ldots, e_{i-1}\} : Y \vdash e_i$.*

We let $\mathbb{F}(\mathcal{E})$ be the set of all the possible configurations of $\mathcal{E}$.

The first condition ensures that a configuration does not contain conflicting events, while the second condition says that for each event e in a configuration there exists a sequence of events $e_1, \ldots, e_n = e$ again in the configuration such that each e_i is enabled by a subset of $\{e_1, \ldots, e_{i-1}\}$. Intuitively, it is thus possible to build a chain of enablings that enables e starting from the empty set and each configuration of a stable event structure can be understood as a computation history up to a certain state.

In our model, we represent tasks in a workflow as events of a stable event structure and we use the terms "event" and "task" interchangeably in the paper, picking the most natural choice based on the context of the discussion. A workflow is a stable event structure including a special event $\checkmark$, representing completion, and extended with a set of constraints C and a task-to-role assignment function ρ. The constraints C allow one to specify that two tasks must be performed by the same user (BoD) or by two different users (SoD), while the function ρ assigns to each task a role which is needed to perform it[1].

Definition 5 (Workflow)**.** *A* workflow *is a triple $\mathcal{W} = \langle \mathcal{E}, C, \rho \rangle$, where:*

1) *$\mathcal{E} = \langle E \cup \{\checkmark\}, \#, \vdash \rangle$ is a stable event structure including an event $\checkmark \notin E$ such that, whenever $X \vdash e$ for some X and e, we have $\checkmark \notin X$;*
2) *$C \subseteq E \times E \times \{=, \neq\}$ is a relation defining a set of BoD and SoD constraints such that:*
   a) *$\forall\, e \in E : (e, e, \neq) \notin C$;*
   b) *$\forall\, e_1, e_2 \in E : (e_1, e_2, =) \in C \Rightarrow (e_2, e_1, =) \in C$;*
   c) *$\forall\, e_1, e_2 \in E : (e_1, e_2, \neq) \in C \Rightarrow (e_2, e_1, \neq) \in C$;*
   d) *$\forall\, e_1, e_2, e_3 \in E : (e_1, e_2, =) \in C \land (e_2, e_3, =) \in C \Rightarrow (e_1, e_3, =) \in C$;*
   e) *$\forall\, e_1, e_2, e_3 \in E : (e_1, e_2, =) \in C \land (e_2, e_3, \neq) \in C \Rightarrow (e_1, e_3, \neq) \in C$;*
3) *$\rho : E \to R$ is a function from events to roles.*

To improve readability, in our examples we do not explicitly close the set of constraints with respect to the rules above. Given a workflow $\mathcal{W} = \langle \langle E, \#, \vdash \rangle, C, \rho \rangle$, we use a subscript notation to extract its different components, e.g., we let $\vdash_{\mathcal{W}}$ stand for $\vdash$ and $C_{\mathcal{W}}$ stand for C.

C. ARBAC + Workflows

Having introduced the ARBAC model and a formal definition of workflow, we now study their interplay by giving a reduction semantics to *ARBAC workflow systems*.

Definition 6 (ARBAC Workflow System)**.** *An* ARBAC workflow system *is a pair $\mathcal{A} = \langle \mathcal{S}, \mathcal{W} \rangle$ including an ARBAC system $\mathcal{S}$ and a workflow $\mathcal{W}$.*

Since workflows in our model allow the specification of BoD and SoD constraints, the reduction semantics of ARBAC workflow systems needs to keep track of the author of the individual tasks. This is formalized by introducing the following notion of *history*.

Definition 7 (History)**.** *Given a workflow $\mathcal{W}$, a* history *$H : E_{\mathcal{W}} \to U$ is a partial function from tasks to users such that $dom(H)$ is a configuration of $\mathcal{E}_{\mathcal{W}}$. We let $\perp$ stand for the empty history.*

Given a history H, we can readily check whether the workflow constraints are satisfied or not. Clearly, a BoD/SoD constraint predicating on the authors of two different tasks can only be checked when the second one is attempted, which leads to the following definition.

[1]This is expressive enough to represent tasks which require multiple roles to be performed or any role in a given set. For instance, if both r_1 and r_2 are needed for a task e, one can introduce in the ARBAC policy a fresh role r which is only granted to users who are assigned both r_1 and r_2. Then, it is enough to let $\rho(e) = r$.

Definition 8 (Satisfiability)**.** *Let $\sim \in \{=, \neq\}$, we say that H* satisfies *the constraint $(e_1, e_2, \sim)$ whenever $H \models (e_1, e_2, \sim)$ can be proved by the following rules:*

$$\frac{e_1 \notin \mathit{dom}(H)}{H \models (e_1, e_2, \sim)} \qquad \frac{e_2 \notin \mathit{dom}(H)}{H \models (e_1, e_2, \sim)} \qquad \frac{H(e_1) \sim H(e_2)}{H \models (e_1, e_2, \sim)}$$

Let $H \models C$ whenever $\forall (e_1, e_2, \sim) \in C : H \models (e_1, e_2, \sim)$.

The operational semantics of an ARBAC workflow system $\mathcal{A} = \langle \mathcal{S}, \mathcal{W} \rangle$ is defined by means of a labelled reduction relation on states $\sigma = \langle UR, H \rangle$, including a user-to-role assignment UR and a history H (see Table II). Labels are drawn from the set of events $E_{\mathcal{W}}$ in the workflow, extended with a distinguished event $\circ$ representing the occurrence of an administrative action (assignment or revocation of roles). To update the history upon reduction, we use the following notation: for a partial function f with $x \notin \mathit{dom}(f)$, let $f[x \mapsto y]$ be the partial function g such that $\mathit{dom}(g) = \mathit{dom}(f) \cup \{x\}$, $g(x) = y$ and $\forall z \in \mathit{dom}(f) : g(z) = f(z)$.

Table II Reduction semantics ($\mathcal{P}, \mathcal{W} \rhd \sigma \xrightarrow{e} \sigma'$)

(R-ADMIN)

$$\frac{\mathcal{P} \rhd UR \rightsquigarrow UR'}{\mathcal{P}, \mathcal{W} \rhd \langle UR, H \rangle \xrightarrow{\circ} \langle UR', H \rangle}$$

(R-TASK)

$$\frac{\begin{array}{cc} \rho_{\mathcal{W}}(e) \in UR(u) & H[e \mapsto u] \models C_{\mathcal{W}} \\ \exists X \subseteq \mathit{dom}(H) : X \vdash_{\mathcal{W}} e & \forall e' \in \mathit{dom}(H) : (e', e) \notin \#_{\mathcal{W}} \end{array}}{\mathcal{P}, \mathcal{W} \rhd \langle UR, H \rangle \xrightarrow{e} \langle UR, H[e \mapsto u] \rangle}$$

Rule (R-ADMIN) is straightforward: it allows one to change the user-to-role assignment in accordance with the underlying ARBAC policy. Rule (R-TASK), instead, models the execution of a task. In words, it is possible to execute a task if: (1) there exists a user who is granted the required role for the task; (2) the execution of the task by this user does not violate the BoD/SoD constraints; (3) the task is enabled by the already performed tasks; and (4) the task does not conflict with any of the already performed tasks.

Observe that the reduction relation in Table II is well-defined, since the premises of rule (R-TASK) ensure that only valid histories are introduced upon reduction when starting from the empty history $\perp$.

III. SECURITY AGAINST COLLUSION

A. Formal Definition of Security

Let $U_c \subseteq U$ be a set of colluding users. Intuitively, an ARBAC workflow system is secure against collusion by U_c whenever no sequence of administrative actions performed by the users in U_c can allow them to complete a workflow which they could not complete just with their original roles. We now formalise this intuition, though we need a number of auxiliary definitions first.

Given a user-to-role assignment UR and a set of colluding users U_c, we let $UR{\downarrow_{U_c}} = \{(u, r) \in UR \mid u \in U_c\}$ stand for the subset of UR including only users in U_c. We extend the notation to ARBAC workflow systems in the expected way, by having $\langle \langle \mathcal{P}, UR \rangle, \mathcal{W} \rangle {\downarrow_{U_c}} = \langle \langle \mathcal{P}, UR{\downarrow_{U_c}} \rangle, \mathcal{W} \rangle$.

Given an ARBAC workflow system $\mathcal{A} = \langle \langle \mathcal{P}, UR \rangle, \mathcal{W} \rangle$, a *trace* of $\mathcal{A}$ is a sequence of events $t = e_1, \ldots, e_n$ such that:

$$\exists \sigma_0, \ldots, \sigma_n : \sigma_0 = \langle UR, \perp \rangle \wedge \forall i \leq n : \mathcal{P}, \mathcal{W} \rhd \sigma_{i-1} \xrightarrow{e_i} \sigma_i.$$

A trace t is *successful* iff ✓ occurs in t. We let $\mathbb{T}^{\checkmark}(\mathcal{A})$ denote the set of the successful traces of $\mathcal{A}$. A trace t is *pure* iff it does not contain any administrative action, i.e., iff $\circ$ does not occur in t. We let $\mathbb{PT}^{\checkmark}(\mathcal{A})$ stand for the set of the pure, successful traces of $\mathcal{A}$.

Definition 9 (Security Against Collusion)**.** *An ARBAC workflow system $\mathcal{A}$ is* secure against collusion by U_c *iff:*

$$\mathbb{T}^{\checkmark}(\mathcal{A}{\downarrow_{U_c}}) \neq \emptyset \Rightarrow \mathbb{PT}^{\checkmark}(\mathcal{A}{\downarrow_{U_c}}) \neq \emptyset.$$

B. Example

We now encode in our formalism the motivating example in Section I-A (Figure 1) and we show that it is not secure against collusion by $\{u_1, u_2\}$.

First, we define $\mathcal{W} = \langle \langle \{a, b, c, \checkmark\}, \vdash, \# \rangle, C, \rho \rangle$, where:

1) the enabling relation is:

$$\vdash = \{(\emptyset, a), (\{a\}, b), (\{a\}, c), (\{b\}, \checkmark), (\{c\}, \checkmark)\}$$

2) the conflict relation is $\# = \{(b, c), (c, b)\}$;
3) $C = \{(a, b, =), (a, c, =)\}$ enforces BoD between tasks a, b and between tasks a, c;
4) $\rho(a) = \mathsf{R}_1, \rho(b) = \mathsf{R}_2, \rho(c) = \mathsf{R}_3$ requires role R_1 to perform task a, role R_2 to perform task b and role R_3 to perform task c.

We then build on top of this workflow the ARBAC workflow system $\mathcal{A} = \langle \langle \mathcal{P}, UR \rangle, \mathcal{W} \rangle$, where:

1) $\mathcal{P} = \langle \{(\mathsf{R}_2, \{\mathsf{R}_1\}, \emptyset, \mathsf{R}_3)\}, \emptyset \rangle$ is the ARBAC policy which allows users with role R_2 to assign role R_3 to any user who is granted role R_1;
2) $UR = \{(u_1, \mathsf{R}_1), (u_2, \mathsf{R}_2)\}$ contains two users u_1, u_2 with roles $\mathsf{R}_1, \mathsf{R}_2$ respectively.

We have that $\mathcal{A}$ is *not* secure against collusion by $\{u_1, u_2\}$ according to Definition 9, since it is possible to put the event ✓ in the history by first assigning role R_3 to u_1 and then letting her execute both a and c; however, without administrative actions (the role assignment by u_2) it is not possible for the two users to complete the workflow.

C. Checking Security Against Collusion

By definition, given an ARBAC workflow system $\mathcal{A}$, its security against collusion by a set of users $U_c \subseteq U$ can be checked as follows:

1) check if there exists a successful trace $t \in \mathbb{T}^{\checkmark}(\mathcal{A}{\downarrow_{U_c}})$: if this is not the case, $\mathcal{A}$ is secure;
2) otherwise, $\mathcal{A}$ is secure if (and only if) there exists also a pure successful trace $t' \in \mathbb{PT}^{\checkmark}(\mathcal{A}{\downarrow_{U_c}})$.

Formally, point (2) amounts to checking the *satisfiability* (or *consistency*) of a workflow, a problem which has received considerable attention in the past [21], [22]. In particular, Wang and Li proved that the workflow satisfiability problem is NP-hard in presence of SoD constraints [22]. Based on this result, it is clear that also point (1) is at least NP-hard in the general case, since points (1) and (2) coincide on the ARBAC workflow system implementing an empty ARBAC policy.

To the best of our knowledge, no algorithm has been proposed so far to deal with point (1). Building all the possible user-to-role assignments for the colluding users based on the ARBAC policy and then checking workflow satisfiability with respect to them is not feasible in practice, given the exponential blow-up of the possible role combinations and the inherent complexity of workflow satisfiability itself. In the next section, we propose the first feasible static analysis technique to solve point (1).

IV. Static Analysis

A. Overview

We propose to check security against collusion in ARBAC workflow systems through a reduction to the well-known *role reachability problem* for ARBAC [17], [20].

Given an ARBAC system $\mathcal{S} = \langle \mathcal{P}, UR \rangle$, a role r is said *reachable* in $\mathcal{S}$ if and only if it can be assigned to some user of the system at some point in time. Formally, this means that there exist a user u and a sequence of user-to-role assignments $UR_0, \ldots, UR_n$ such that $UR_0 = UR$ and:

$$(\forall i \leq n : \mathcal{P} \rhd UR_{i-1} \rightsquigarrow UR_i) \wedge r \in UR_n(u).$$

We propose to encode the workflow as a set of can-assign rules, which extend the original ARBAC policy $\mathcal{P}$ so that a specific role (introduced by the encoding) is reachable *if and only if* the workflow can be completed by the colluding users, possibly by making use of administrative actions; this characterization is proved correct for a restricted, yet useful, class of worklow systems discussed below. Notably, however, even for workflow systems which do not belong to this class, we prove that the unreachability of the role above ensures that the workflow cannot be completed by the colluding users, which may be enough to prove security against collusion in many practical cases.

By internalizing the workflow into the underlying ARBAC policy, we can reuse efficient tools for role reachability analysis [13], [9], [16] to prove or disprove security against collusion in workflow systems. Moreover, having a unified representation (in terms of ARBAC) of both the ARBAC policy and the workflow to analyse makes it possible to devise aggressive optimizations which exploit as much information as possible to simplify the security problem and make it tractable.

B. Reduction to Role Reachability

Let $\mathcal{A} = \langle \mathcal{S}, \mathcal{W} \rangle$, for each task $e \in E_{\mathcal{W}}$ we introduce three fresh roles: $\mathsf{Allowed}[e]$, $\mathsf{Author}[e]$ and $\mathsf{Done}[e]$. Moreover, for each pair of tasks e_1, e_2 such that $(e_1, e_2, =) \in C_{\mathcal{W}}$, we introduce a fresh role $\mathsf{Eq}[e_1, e_2]$. Finally, we introduce a fresh role Super, which will always be assigned to a dummy user introduced by the encoding. Notice that we can always ensure that the set of roles in $\mathcal{A}$ does not clash with the set of roles introduced by the encoding just by performing a preliminary renaming of the elements of the former.

The core of the encoding is a translation from a workflow $\mathcal{W}$ into a set of can-assign rules $[\![\mathcal{W}]\!]$. The translation requires the generation of the set of the configurations of $\mathcal{E}_{\mathcal{W}}$ and it assumes the following definition of *precedence*, which can be used to identify the set of the predecessors of an event in a stable event structure [18].

Definition 10 (Precedence). *For a stable event structure $\mathcal{E} = \langle E, \vdash, \# \rangle$ and a configuration $X \in \mathbb{F}(\mathcal{E})$, we define the* precedence *relation $\prec_X \subseteq X \times X$ by having $e \prec_X e'$ if and only if $\exists\, Y \subseteq X : e \in Y \wedge Y \vdash e'$. We then let $<_X$ stand for the transitive closure of $\prec_X$.*

We let $[\![\mathcal{W}]\!]$ be the smallest set of can-assign rules derived by the inference rules in Table III. The core intuitions underlying the translation can be summarized as follows:

- we assume the existence of a dummy user with the Super role, which we call the *super user*. We use the super user to trigger several can-assign rules generated by the translation and to keep track in the encoding of which tasks can be executed or have been performed so far. This is done by assigning to the super user a specific set of roles (again introduced by the translation) and by ensuring that they satisfy the invariants explained below;
- rule (T-Allowed): to assign role $\mathsf{Allowed}[e]$, we must ensure that: (1) all the predecessors of e have already been performed; (2) none of the predecessors of e violates a BoD constraint; and (3) no event which is conflicting with e has been previously allowed. Notice that $\mathsf{Allowed}[e]$ can only be assigned to the super user: this ensures that all the information about the allowed events is centralized on the super user, i.e., in the encoding we can always be aware of all the tasks which have been allowed so far;
- rule (T-Author): once $\mathsf{Allowed}[e]$ has been assigned to the super user, it is possible to attempt the assignment of role $\mathsf{Author}[e]$. This role can only be assigned to a user who has the required role to perform e according to the task-to-role assignment function ρ; moreover, the user must satisfy all the BoD constraints between e and its predecessors, as well as all the SoD constraints between e and the other tasks of the workflow. Observe that the super user can never be assigned $\mathsf{Author}[e]$, reflecting the intuition that he is just a dummy user introduced by the encoding and not a real user of the system;
- rule (T-Done): once $\mathsf{Author}[e]$ has been assigned to some user, it is possible to assign role $\mathsf{Done}[e]$ to the super user. This role assignment tracks that it was indeed possible to perform task e. Role $\mathsf{Done}[e]$ may be required to enable the assignment of role $\mathsf{Allowed}[e']$ for some event e' which is a successor of e;
- rule (T-Eq): if a user is assigned both $\mathsf{Author}[e_1]$ and $\mathsf{Author}[e_2]$, she can also be assigned role $\mathsf{Eq}[e_1, e_2]$, thus

Table III Translation of a workflow $\mathcal{W}$ into a set of can-assign rules $[\![\mathcal{W}]\!]$

(T-ALLOWED)
$$\frac{X \in \mathbb{F}(\mathcal{E}_{\mathcal{W}}) \qquad e \in X \qquad \mathit{Done} = \{\mathsf{Done}[e'] \mid e' <_X e\} \qquad \mathit{Eqs} = \{\mathsf{Eq}[e_1, e_2] \mid e_1 <_X e \wedge e_2 <_X e \wedge (e_1, e_2, =) \in C_{\mathcal{W}}\}}{(\mathsf{Super}, \mathit{Done} \cup \mathit{Eqs} \cup \{\mathsf{Super}\}, \{\mathsf{Allowed}[e'] \mid (e, e') \in \#_{\mathcal{W}}\}, \mathsf{Allowed}[e]) \in [\![\mathcal{W}]\!]}$$

(T-EQ)
$$\frac{(e_1, e_2, =) \in C_{\mathcal{W}}}{(\mathsf{Super}, \{\mathsf{Author}[e_1], \mathsf{Author}[e_2]\}, \emptyset, \mathsf{Eq}[e_1, e_2]) \in [\![\mathcal{W}]\!]}$$

(T-PROPEQ)
$$\frac{(e_1, e_2, =) \in C_{\mathcal{W}}}{(\mathsf{Eq}[e_1, e_2], \{\mathsf{Super}\}, \emptyset, \mathsf{Eq}[e_1, e_2]) \in [\![\mathcal{W}]\!]}$$

(T-AUTHOR)
$$\frac{X \in \mathbb{F}(\mathcal{E}_{\mathcal{W}}) \qquad e \in X \qquad \mathit{Neqs} = \{\mathsf{Author}[e'] \mid (e', e, \neq) \in C_{\mathcal{W}}\} \qquad \mathit{Eqs} = \{\mathsf{Author}[e'] \mid e' <_X e \wedge (e', e, =) \in C_{\mathcal{W}}\}}{(\mathsf{Allowed}[e], \{\rho_{\mathcal{W}}(e)\} \cup \mathit{Eqs}, \mathit{Neqs} \cup \{\mathsf{Super}\}, \mathsf{Author}[e]) \in [\![\mathcal{W}]\!]}$$

(T-DONE)
$$\frac{e \in E_{\mathcal{W}}}{(\mathsf{Author}[e], \{\mathsf{Super}\}, \emptyset, \mathsf{Done}[e]) \in [\![\mathcal{W}]\!]}$$

proving that a BoD constraint between e_1 and e_2 has been satisfied. Afterwards, by rule (T-PROPEQ), role $\mathsf{Eq}[e_1, e_2]$ can be further assigned to the super user: this may be needed to enable the assignment of role $\mathsf{Allowed}[e]$ for some event e following both e_1 and e_2.

Finally, the translation is extended so as to map an ARBAC workflow system $\mathcal{A} = \langle\langle\langle CA, CR\rangle, UR\rangle, \mathcal{W}\rangle$ into a corresponding ARBAC system $[\![\mathcal{A}]\!]$ as follows:

$$[\![\mathcal{A}]\!] = \langle\langle CA^- \cup [\![\mathcal{W}]\!], CR\rangle, UR \cup \{(u_0, \mathsf{Super})\}\rangle,$$

where u_0 is a fresh user extending the set of users U and CA^- is the set of the can-assign rules obtained by including Super in the negative preconditions of all the rules in CA.

C. Formal Results

The first result we prove for the encoding we detailed is the following soundness theorem.

Theorem 1 (Soundness). *For any $\mathcal{A}$ such that $\mathbb{T}^{\checkmark}(\mathcal{A}) \neq \emptyset$, the role* $\mathsf{Done}[\checkmark]$ *is reachable in* $[\![\mathcal{A}]\!]$.

Proof. See the long version [10]. □

The soundness theorem ensures that, if role $\mathsf{Done}[\checkmark]$ is not reachable in $[\![\mathcal{A}]\!]$, then the ARBAC workflow system $\mathcal{A}$ will never reach a state where the workflow has been completed by the system users, even by making use of administrative actions. In terms of security this means that, given a set of colluding users U_c, the unreachability of role $\mathsf{Done}[\checkmark]$ in $[\![\mathcal{A}\downarrow_{U_c}]\!]$ ensures that the users in U_c cannot complete the workflow, not even by changing roles, hence we get a proof of security for $\mathcal{A}$ against collusion by U_c (by Definition 9).

Interestingly, we are also able to establish a completeness theorem for the restricted case of ARBAC workflow systems without BoD constraints.

Theorem 2 (Completeness). *For any $\mathcal{A}$ not including BoD constraints, if the role* $\mathsf{Done}[\checkmark]$ *is reachable in* $[\![\mathcal{A}]\!]$*, we have that $\mathbb{T}^{\checkmark}(\mathcal{A}) \neq \emptyset$.*

Proof. See the long version [10]. □

The completeness theorem is useful to confirm the presence of a collusion attack. Let $\mathcal{A}$ be an ARBAC workflow system and U_c be a set of colluding users, and assume we proved, for instance by using the techniques in [21], [22], that $\mathcal{A}\downarrow_{U_c}$ does not allow the users in U_c to complete the workflow just with their original roles. By Theorem 2, if we show that role $\mathsf{Done}[\checkmark]$ is reachable in $[\![\mathcal{A}\downarrow_{U_c}]\!]$ and the workflow does not include any BoD constraint, we can confirm that the users in U_c can complete the workflow by making use of administrative actions, hence $\mathcal{A}$ is not secure against collusion by U_c (again by Definition 9).

We conjecture that the completeness theorem can be actually extended to arbitrary ARBAC workflow systems, but this stronger property looks significantly harder to prove than Theorem 2. We provide an intuition on why excluding BoD constraints helps in the proof of Theorem 2. One of the subtlest differences between ARBAC systems and ARBAC workflow systems is that in the former any role, including the roles of the form $\mathsf{Author}[e]$ for some task e, can be assigned to all users satisfying the preconditions of the can-assign rules granting that role; conversely, a task can be performed only once in an ARBAC workflow system, so the author of each task is uniquely determined there. Thus, in the encoding into ARBAC, a role like $\mathsf{Author}[e]$ can be potentially assigned to multiple users, though only one of these users actually performs e in the ARBAC workflow system. To prove completeness, one needs to show that assigning $\mathsf{Author}[e]$ to multiple users does not introduce "false attacks". Notably, if no BoD constraint is put in place in the workflow system, roles like $\mathsf{Author}[e]$ do not occur in any positive precondition of the can-assign rules generated by the encoding into ARBAC: this means that it is enough to have just one user who is assigned the role, to trigger those can-assign rules where $\mathsf{Author}[e]$ is in administrative position (i.e., it appears as the first element of the rule). Without loss of generality, it is thus possible in the proof of Theorem 2 to only focus on *well-formed* traces, where each role like $\mathsf{Author}[e]$ is assigned at most once.

It should not be surprising that the encoding we propose can also be used to solve the classic workflow satisfiability problem [21], [22] in the case of ARBAC workflow systems without BoD constraints. Given $\mathcal{A} = \langle\langle\mathcal{P}, \mathit{UR}\rangle, \mathcal{W}\rangle$, let:

$$[\![\mathcal{A}]\!]^* = \langle\langle[\![\mathcal{W}]\!], \emptyset\rangle, \mathit{UR} \cup \{(u_0, \mathsf{Super})\}\rangle,$$

where u_0 is a fresh user extending the set of users U.

Lemma 1 (Workflow Satisfiability)**.** *For any $\mathcal{A}$, $\mathbb{PT}^{\checkmark}(\mathcal{A}) \neq \emptyset$ implies that the role $\mathsf{Done}[\checkmark]$ is reachable in $[\![\mathcal{A}]\!]^*$. Moreover, if $\mathcal{A}$ does not include BoD constraints, then also the converse holds true.*

Proof. It is enough to observe that $\mathbb{PT}^{\checkmark}(\langle\langle\mathcal{P}, \mathit{UR}\rangle, \mathcal{W}\rangle) = \mathbb{T}^{\checkmark}(\langle\langle\mathcal{P}_\perp, \mathit{UR}\rangle, \mathcal{W}\rangle)$, where $\mathcal{P}_\perp = \langle\emptyset, \emptyset\rangle$ is the empty ARBAC policy. The result then is an immediate consequence of the theorems above. □

In terms of computational complexity, one can observe that the workflow satisfiability problem is NP-hard in presence of SoD constraints [22], which we consider in this work. The encoding $[\![\cdot]\!]^*$ generates an ARBAC system without can-revoke rules, so the corresponding role reachability problem is NP-complete [17]. Hence, in the general case, both problems are equally hard, which in principle makes our approach a viable solution also for the classic workflow satisfiability problem.

We conclude this section by summarizing in Table IV how Theorem 1 and Theorem 2 can be used to check the security of an ARBAC workflow system $\mathcal{A}$ against collusion by U_c. Notice that, if $\mathsf{Done}[\checkmark]$ is not reachable in $[\![\mathcal{A}{\downarrow_{U_c}}]\!]$, we can immediately prove security and ignore $[\![\mathcal{A}{\downarrow_{U_c}}]\!]^*$. One case is missing from the table, since it is not possible that $\mathsf{Done}[\checkmark]$ is reachable in $[\![\mathcal{A}{\downarrow_{U_c}}]\!]^*$, but not in $[\![\mathcal{A}{\downarrow_{U_c}}]\!]$.

Table IV Checking security of $\mathcal{A}$ against collusion by U_c

BoD	Reach in $[\![\mathcal{A}{\downarrow_{U_c}}]\!]$	Reach in $[\![\mathcal{A}{\downarrow_{U_c}}]\!]^*$	Secure
no	no	no	yes
no	yes	no	no
no	yes	yes	yes
yes	no	no	yes
yes	yes	no	?
yes	yes	yes	?

D. Optimizations

The encoding of ARBAC workflow systems into ARBAC we presented enjoys important formal properties, but it may lead to the generation of large and complex ARBAC systems, which do not admit an efficient role reachability analysis. Luckily, there is a well-established approach to make role reachability tractable for ARBAC, that is the use of *pruning* techniques, which perform syntactic transformations shrinking the size of the analysed ARBAC system, without affecting the reachability of a given role of interest [17], [15], [13].

We present here an effective pruning technique we devised to simplify the role reachability problems generated by our encoding: though the pruning has been designed to work at its best in this specific context, we expect several ideas to be general enough to be useful for the simplification of arbitrary ARBAC systems.

1) Definitions: A role r is *administrative* if and only if there exists at least one can-assign rule of the form (r, R_p, R_n, r_t) for some R_p, R_n, r_t or one can-revoke rule of the form (r, r') for some r'. A role is *positive* (resp. *negative*) if and only if it occurs in the positive (resp. negative) preconditions of some can-assign rule [13]. We say that a role is *purely administrative* iff it is administrative, non-positive and non-negative. Purely administrative roles only need to be assigned to grant the right of performing some administrative actions. As such, it is not important to know who is granted a purely administrative role, as long as there is one user having the role when the administrative actions it enables are intended to be performed.

A role is *positively stable* if and only if it is irrevocable or non-negative. A role is *negatively stable* if and only if it is not assignable or it is both non-positive and non-administrative. If a user is assigned a positively stable role r, we can assume that r will be assigned to her forever, either because it cannot be removed (if r is irrevocable) or because removing it does not enable new administrative actions (if r is non-negative). Dually, if a user is not assigned a negatively stable role r', we can assume that r' will never be assigned to her.

2) Preprocessing: Before pruning, each rule of the form $(\mathsf{Super}, R_p, R_n, \mathsf{Allowed}[e])$ is replaced by the set of rules:

$$\{(\mathsf{Author}[e'], R_p, R_n, \mathsf{Allowed}[e]) \mid \mathsf{Done}[e'] \in R_p\}.$$

It can be shown that this preprocessing step does not affect the reachability of any role, since each rule of the previous format is enabled if and only if all the rules in the set generated from it are enabled. The intuition behind this observation is that, for each event e', role $\mathsf{Done}[e']$ can be assigned if and only if role $\mathsf{Author}[e']$ is first assigned.

Moreover, as part of the preprocessing step, all roles of the form $\mathsf{Allowed}[e]$ are removed from the negative preconditions of the can-assign rules generated by the encoding. Though this may look surprising, it can be shown that it does not affect the reachability of role $\mathsf{Done}[\checkmark]$, since the negative preconditions of the previous form are just an artefact to simplify the proof of the main formal results. To understand why the reachability of $\mathsf{Done}[\checkmark]$ is not affected by this change, observe that a role like $\mathsf{Allowed}[e]$ is only included in the negative preconditions of a can-assign rule generated by the encoding in Table III if e is conflicting with some other event. Assume then there are two conflicting events e_1, e_2: according to the encoding, only one role between $\mathsf{Author}[e_1]$ and $\mathsf{Author}[e_2]$ can be assigned, since only one between $\mathsf{Allowed}[e_1]$ and $\mathsf{Allowed}[e_2]$ can be given to the super user. However, assume that both $\mathsf{Author}[e_1]$ and $\mathsf{Author}[e_2]$ were assigned to some user, since we removed the negative preconditions above. This may lead to two potentially troublesome scenarios, which would never happen if only one of the two roles was assigned:

1) $\mathsf{Done}[e_1]$ and $\mathsf{Done}[e_2]$ get both assigned;
2) $\mathsf{Eq}[e_1, e_2]$ gets assigned (if $\mathsf{Author}[e_1]$ and $\mathsf{Author}[e_2]$ are given to the same user).

Both these cases do not affect the reachability of $\mathsf{Done}[\checkmark]$, since e_1 and e_2 are conflicting and thus never included in the same configuration of the stable event structure underlying the workflow. By the definition of the encoding, this implies that $\mathsf{Done}[e_1]$ and $\mathsf{Done}[e_2]$ never occur together in the positive preconditions of a can-assign rule, and similarly $\mathsf{Eq}[e_1, e_2]$ is not included in any positive precondition, so the cases above do not enable more administrative actions.

Though the preprocessing we discussed increases the number of can-assign rules generated by the encoding, the newly introduced rules have a format which is more amenable for pruning and, pragmatically, eventually leads to the generation of ARBAC systems which are smaller and easier to analyse. Alternatively, one could skip the preprocessing and fine-tune the pruning rules to improve their effectiveness, but this would make the rules harder to present.

3) Pruning Rules: The pruning rules make use of a binary relation on roles $\preceq$. Intuitively, $r \preceq r'$ ensures that, whenever r' is assigned to some user of an ARBAC system, then also r is assigned to some user (not necessarily the same one). Formally, $\preceq$ is defined as the least pre-order satisfying the following two clauses:

1) $r \preceq r'$ for all r' if r is initially assigned and positively stable;
2) $r \preceq r'$ if r is positively stable, r' is initially unassigned, and for all can-assign rules of the form (r_a, R_p, R_n, r') we have $r \in R_p \cup \{r_a\}$.

We are finally ready to present the pruning rules, assuming an initial user-to-role assignment UR:

1) Let r_t be a non-negative role. If there exist a rule $ca = (r, R_p, R_n, r_t)$ and a rule $ca' = (r', R'_p \cup \{r_t\}, R'_n, r'_t)$ with $R_p \subseteq R'_p$, $R_n \subseteq R'_n$ and there exists $r'' \in R'_p \cup \{r'\}$ such that $r \preceq r''$, then replace ca' with (r', R'_p, R'_n, r'_t);
2) Let r_t be a role. If there exist a rule $ca = (r, R_p, R_n, r_t)$ and a rule $ca' = (r', R'_p, R'_n, r_t)$ with $R_p \subseteq R'_p$, $R_n \subseteq R'_n$ and there exists $r'' \in R'_p \cup \{r'\}$ such that $r \preceq r''$, then remove ca';
3) Let r_t be a purely administrative role and let $r \preceq r_t$. If there exist a rule $ca = (r, R_p, R_n, r_t)$ and a user u such that $R_p \subseteq UR(u)$ and $R_n \cap UR(u) = \emptyset$, the roles in R_p are positively stable and the roles in R_n are negatively stable, then remove ca and replace all the occurrences of r_t with r in the can-assign/can-revoke rules.

Rule 1 says that, if there exist a rule ca assigning a non-negative role r_t and a rule ca' including r_t in the positive preconditions, we can drop r_t from the positive preconditions of ca', as long as we are guaranteed that ca is always enabled when ca' is enabled up to the absence of r_t. Rule 2 says that, if we have two rules ca and ca' assigning the same role r_t and ca is always enabled when ca' is enabled, we can remove rule ca'. These rules are reminiscent of two pruning rules originally presented in [15], noted there as rules R_2 and R_5 respectively, but they make use of the $\preceq$ relation to be more general and more effective on our cases.

Rule 3 deals with the assignment of purely administrative roles and it is trickier: it allows one to delegate the administrative rights of a purely administrative role r_t to any $r \preceq r_t$. Notice that this change always preserves role reachability, since whenever r_t is assigned to some user, also r must be assigned to someone by definition of the $\preceq$ relation. However, this change could alter the semantics of the ARBAC system if r is assigned, but r_t is not immediately assignable. To ensure that this does not happen, we have to check a few additional conditions. In particular, rule 3 checks the existence of a can-assign rule $ca = (r, R_p, R_n, r_t)$ whose preconditions are satisfied by some user u in the initial user-to-role assignment: if the roles in R_p are positively stable and the roles in R_n are negatively stable, we can assume that the preconditions of ca will always be satisfied by u, hence ensuring that r_t is always assignable when r is assigned to someone.

The pruning algorithm just amounts to continuously applying rules 1-3 to the ARBAC system, until no more rules can be applied. Termination is ensured by the observation that all rules reduce the size of the ARBAC system, either in terms of the number of can-assign/can-revoke rules or in terms of the size of the positive preconditions of the can-assign rules.

The pruning algorithm enjoys the following property:

Theorem 3. *Role* $\mathsf{Done}[\checkmark]$ *is reachable in* $[\![\mathcal{A}]\!]$ *if and only if it is reachable after pruning* $[\![\mathcal{A}]\!]$.

Proof. See the long version [10]. □

V. Examples

We show the static analysis at work on some simple, but representative examples. To improve readability, we only present the most interesting subset of the can-assign rules generated by the encoding into ARBAC. We do not apply the pruning algorithm to these simple examples, but we just present the result of the direct application of the encoding.

A. Exclusive Choice

Consider the workflow described in the motivating example in Section I-A (Figure 1), its translation into ARBAC is given in Table V.

Notice that roles $\mathsf{Allowed}[b]$ and $\mathsf{Allowed}[c]$ are mutually exclusive, since events b and c are conflicting: this implies that only one role between $\mathsf{Author}[b]$ and $\mathsf{Author}[c]$ can be assigned. Correspondingly, there are two ways to introduce role $\mathsf{Allowed}[\checkmark]$: indeed, recall that role $\mathsf{Done}[b]$ / $\mathsf{Done}[c]$ can only be assigned by a user with role $\mathsf{Author}[b]$ / $\mathsf{Author}[c]$. Moreover, notice that both $\mathsf{Author}[b]$ and $\mathsf{Author}[c]$ can only be assigned to a user who is assigned $\mathsf{Author}[a]$, thus ensuring that the BoD constraints in the workflow are satisfied. Finally, observe that the roles required to perform a, b, c according to the task-to-role assignment function of the workflow are included in the positive preconditions of the rules assigning the corresponding author role.

Table V Translation of the workflow in Figure 1

$$(\mathsf{Super}, \{\mathsf{Super}\}, \emptyset, \mathsf{Allowed}[a])$$

$$(\mathsf{Super}, \{\mathsf{Done}[a], \mathsf{Super}\}, \{\mathsf{Allowed}[c]\}, \mathsf{Allowed}[b])$$

$$(\mathsf{Super}, \{\mathsf{Done}[a], \mathsf{Super}\}, \{\mathsf{Allowed}[b]\}, \mathsf{Allowed}[c])$$

$$(\mathsf{Super}, \{\mathsf{Done}[a], \mathsf{Done}[b], \mathsf{Eq}[a,b], \mathsf{Super}\}, \emptyset, \mathsf{Allowed}[\checkmark])$$

$$(\mathsf{Super}, \{\mathsf{Done}[a], \mathsf{Done}[c], \mathsf{Eq}[a,c], \mathsf{Super}\}, \emptyset, \mathsf{Allowed}[\checkmark])$$

$$(\mathsf{Super}, \{\mathsf{Author}[a], \mathsf{Author}[b]\}, \emptyset, \mathsf{Eq}[a,b])$$

$$(\mathsf{Super}, \{\mathsf{Author}[a], \mathsf{Author}[c]\}, \emptyset, \mathsf{Eq}[a,c])$$

$$(\mathsf{Eq}[a,b], \{\mathsf{Super}\}, \emptyset, \mathsf{Eq}[a,b])$$

$$(\mathsf{Eq}[a,c], \{\mathsf{Super}\}, \emptyset, \mathsf{Eq}[a,c])$$

$$(\mathsf{Allowed}[a], \{\mathsf{R}_1\}, \{\mathsf{Super}\}, \mathsf{Author}[a])$$

$$(\mathsf{Allowed}[b], \{\mathsf{R}_2, \mathsf{Author}[a]\}, \{\mathsf{Super}\}, \mathsf{Author}[b])$$

$$(\mathsf{Allowed}[c], \{\mathsf{R}_3, \mathsf{Author}[a]\}, \{\mathsf{Super}\}, \mathsf{Author}[c])$$

$$(\mathsf{Allowed}[\checkmark], \emptyset, \{\mathsf{Super}\}, \mathsf{Author}[\checkmark])$$

Table VI Translation of the workflow in Figure 2

$$(\mathsf{Super}, \{\mathsf{Super}\}, \emptyset, \mathsf{Allowed}[a])$$

$$(\mathsf{Super}, \{\mathsf{Done}[a], \mathsf{Super}\}, \emptyset, \mathsf{Allowed}[b])$$

$$(\mathsf{Super}, \{\mathsf{Done}[a], \mathsf{Done}[b], \mathsf{Super}\}, \emptyset, \mathsf{Allowed}[c])$$

$$(\mathsf{Super}, \{\mathsf{Done}[a], \mathsf{Done}[b], \mathsf{Done}[c], \mathsf{Super}\}, \emptyset, \mathsf{Allowed}[\checkmark])$$

$$(\mathsf{Allowed}[a], \{\mathsf{R}_1\}, \{\mathsf{Author}[c], \mathsf{Super}\}, \mathsf{Author}[a])$$

$$(\mathsf{Allowed}[b], \emptyset, \{\mathsf{Super}\}, \mathsf{Author}[b])$$

$$(\mathsf{Allowed}[c], \{\mathsf{R}_2\}, \{\mathsf{Author}[a], \mathsf{Super}\}, \mathsf{Author}[c])$$

$$(\mathsf{Allowed}[\checkmark], \emptyset, \{\mathsf{Super}\}, \mathsf{Author}[\checkmark])$$

B. Sequential Execution

Consider the workflow in Figure 2, including three sequential tasks a, b, c such that a and c must be performed by different authors. Its translation into ARBAC is given in Table VI.

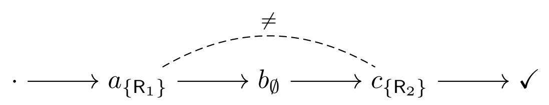

Figure 2. Sequential execution with separation-of-duty

Observe that role $\mathsf{Author}[c]$ can only be assigned to a user who is not assigned $\mathsf{Author}[a]$ and vice-versa, thus ensuring that the SoD constraint between a and c is satisfied.

C. Parallel Execution

Consider the workflow in Figure 3, including two parallel tasks a, b, which must both be performed before completing the workflow, graphically noted by joining the two edges entering $\checkmark$ (formally, this can be represented by the enabling relation $\emptyset \vdash a$, $\emptyset \vdash b$ and $\{a, b\} \vdash \checkmark$). Moreover, assume there exists a BoD constraint between a and b.

The translation of the workflow into ARBAC is shown in Table VII. Since there is no temporal dependence between a and b, it is not possible to predict which of the two tasks

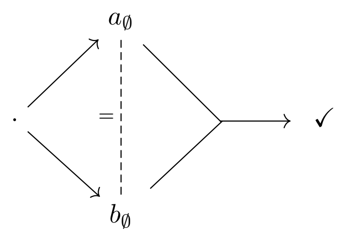

Figure 3. Parallel execution with binding-of-duty

is executed before: hence, both $\mathsf{Allowed}[a]$ and $\mathsf{Allowed}[b]$, as well as $\mathsf{Author}[a]$ and $\mathsf{Author}[b]$, can be liberally assigned without checking the BoD constraint between a and b. However, $\mathsf{Allowed}[\checkmark]$ can only be introduced whenever $\mathsf{Eq}[a, b]$ is assigned to the super user, which is only possible when there exists a user who is assigned both $\mathsf{Author}[a]$ and $\mathsf{Author}[b]$, i.e., when the BoD constraint between a and b is satisfied.

VI. Implementation

We developed WARBAC, a tool for checking the security against collusion of ARBAC workflow systems. Given an input file including the specification of an ARBAC workflow system $\mathcal{A}$ and a set of colluding users U_c, it runs a security verification by: (1) removing from $\mathcal{A}$ all the users not included in U_c; (2) encoding the security problem for the resulting system in terms of role reachability for ARBAC, based on the presented theory; and (3) simplifying the role reachability problem by running the pruning algorithm in Section IV-D. The generated role reachability queries for role $\mathsf{Done}[\checkmark]$ are then discharged by an existing state-of-the-art tool, VAC [13].

A. Implementing the Analysis

WARBAC reduces security against collusion to role reachability by implementing the checks summarised in Table IV (in Section IV-C). Since role reachability may be costly to check,

Table VII Translation of the workflow in Figure 3

$$(\mathsf{Super}, \{\mathsf{Super}\}, \emptyset, \mathsf{Allowed}[a])$$

$$(\mathsf{Super}, \{\mathsf{Super}\}, \emptyset, \mathsf{Allowed}[b])$$

$$(\mathsf{Super}, \{\mathsf{Done}[a], \mathsf{Done}[b], \mathsf{Eq}[a,b], \mathsf{Super}\}, \emptyset, \mathsf{Allowed}[\checkmark])$$

$$(\mathsf{Super}, \{\mathsf{Author}[a], \mathsf{Author}[b]\}, \emptyset, \mathsf{Eq}[a,b])$$

$$(\mathsf{Eq}[a,b], \{\mathsf{Super}\}, \emptyset, \mathsf{Eq}[a,b])$$

$$(\mathsf{Allowed}[a], \emptyset, \{\mathsf{Super}\}, \mathsf{Author}[a])$$

$$(\mathsf{Allowed}[b], \emptyset, \{\mathsf{Super}\}, \mathsf{Author}[b])$$

$$(\mathsf{Allowed}[\checkmark], \{\mathsf{Super}\}, \emptyset, \mathsf{Author}[\checkmark])$$

WARBAC exploits different analysis backends supported by VAC to make the security analysis more efficient:

1) INTERPROC [2]: efficient, sound, but over-approximated analysis. A negative answer by INTERPROC proves role unreachability, but a positive answer may be incorrectly returned as the result of an over-approximation [14];
2) CBMC [1]: efficient bounded model-checking. A positive answer by CBMC proves role reachability, but a negative answer may be incorrectly returned as the result of a bound on the search space [13];
3) NuSMV [3]: computationally expensive sound and complete analysis. A positive/negative answer by NuSMV proves role reachability/unreachability [13].

When testing role reachability, WARBAC first tries to prove unreachability by using INTERPROC; if this fails, it attempts to prove reachability by using CBMC with a depth search empirically set to 14; if this also fails, it resorts to running NuSMV to get a final answer.

Moreover, WARBAC tries to simplify as much as possible the generated role reachability problems before attempting to solve them. In particular, it applies the following procedure: 1) run the pruning algorithm described in Section IV-D, and 2) run the pruning algorithm internally implemented in VAC, until no further simplification is possible.

B. Experiments

We created a set of examples to test WARBAC, which we make available online [10]. All the examples refer to a medical setting: specifically, we extended an existing ARBAC system (the Hospital case study in [13]) with the specification of a number of different workflows. Most of the workflows we developed are larger, more complicated variants of the examples shown in Section V, implementing different patterns: sequential execution, parallel execution and exclusive choice. We also developed more complex workflows, representing realistic first-aid procedures. All the experiments were performed on a 64-bit Intel Xeon running at 2.4 GHz.

1) Synthetic Examples: Table VIII reports on the experimental results. The table shows for each example the following information:

1) the main pattern underlying the workflow, e.g., sequential;
2) the number of tasks in the workflow;
3) the type of enforced constraints (BoD or SoD);
4) the number of colluding users originally in input, after the pruning implemented in VAC is enabled, and after the full pruning is enabled;
5) the number of can-assign/can-revoke rules originally in input, after the pruning implemented in VAC is enabled, and after the full pruning is enabled;
6) the aggregate analysis time when only the pruning implemented in VAC is enabled and when the full pruning is enabled (the analysis never terminates within one hour if no form of pruning is enabled, so we do not report this information);
7) the expected analysis result (safe or unsafe) and the answer reported by WARBAC.

The first obvious observation from the table is that enabling the full pruning algorithm is very important for the scalability of the analysis: in 13 out of 21 examples the improvement in performances is dramatic, with 7 cases failing to terminate within one hour if only the internal pruning of VAC is enabled, but analysed in a few minutes if the full pruning is used. There are 8 cases where activating the full pruning turns out to be overshooting, since the pruning performed by VAC is already very effective and the additional overhead of running the full pruning is not justified. Still, all these cases can be solved in seconds in both scenarios.

Most of the safe cases required WARBAC to only run INTERPROC. As expected, the over-approximated analysis implemented in INTERPROC is very fast, since it only takes a few seconds in all the test cases, even the most complicated ones. Though over-approximated, the analysis performed by INTERPROC is useful in many practical cases, e.g., when it finds that a role required to complete the workflow is neither assigned initially, nor assignable to any of the colluding users. When INTERPROC is not able to prove security, WARBAC runs CBMC and possibly NuSMV to get a more precise answer. Though the analysis implemented in NuSMV is potentially costly, our optimization techniques proved very effective to provide good analysis times, even for large settings.

We find it promising that realistic cases like the first-aid procedures described in the next section are analysed in minutes, though in some cases WARBAC is unable to prove security, since our analysis is not proved complete for workflows using BoD constraints.

2) Case Study: The main case study we considered in our experiments is a workflow modelling a procedure to assist a patient in need for a first aid treatment. The workflow includes 10 different tasks:

a) a patient comes at the hospital and gets a ticket;

Table VIII Experimental results

			Colluding Users			Rules			Aggregate Time		Results	
Type	Tasks	Cons	Orig	VAC	FullPrune	Orig	VAC	FullPrune	VAC	FullPrune	Expected	Answer
sequential	9	SoD	121	74	47	55	37	23	>60m	2m41s	U	U
sequential	9	BoD	121	76	46	57	39	22	>60m	2m31s	U	?
sequential	9	SoD	121	5	3	55	2	1	6s	20s	S	S
sequential	12	SoD	121	86	53	64	46	26	>60m	4m36s	U	U
sequential	12	BoD	121	88	25	66	48	17	>60m	1m45s	U	?
sequential	12	SoD	121	5	4	64	2	1	5s	56s	S	S
parallel	9	SoD	121	74	47	55	37	23	>60m	2m39s	U	U
parallel	9	BoD	121	6	3	57	3	1	4s	22s	S	S
parallel	9	SoD	121	5	3	55	2	1	4s	22s	S	S
parallel	12	SoD	121	88	28	64	46	17	>60m	4m05s	S	S
parallel	12	BoD	121	88	25	66	48	17	53m52s	1m46s	U	?
parallel	12	BoD	121	6	3	66	3	1	5s	51s	S	S
choice	9	SoD	121	28	21	57	41	28	20m56s	5m55s	U	U
choice	9	SoD	121	64	46	57	32	19	5m1s	1m12s	S	S
choice	9	SoD	121	4	3	57	2	1	4s	20s	S	S
choice	12	SoD	121	42	33	66	52	36	>60m	17m15s	U	U
choice	12	SoD	121	89	50	66	48	25	42m02s	2m53s	S	S
choice	12	SoD	121	4	3	66	2	1	5s	40s	S	S
first-aid	10	SoD	121	30	20	65	49	37	35m17s	12m09s	U	U
first-aid	10	SoD	121	50	34	65	49	37	43m15s	15m32s	S	S
first-aid	10	SoD	121	5	3	65	2	1	5s	29s	S	S

$b)$ a doctor makes a preliminary evaluation and sends the patient in for a visit (f), while she provides the relative documentation to a receptionist (c);

$c)$ a receptionist makes the paperwork summarizing the conditions of the patient, possibly while her visits are still ongoing;

$d)$ when the paperworks are ready (c done) and the patient has been dismissed (i done), a receptionist closes the patient case suggesting additional treatment (j) or not (e);

$e)$ the patient is fine: she shows the paperwork at the exit and leaves the hospital (✓);

$f)$ after the preliminary evaluation, a nurse marks the case as urgent (g) or not (h);

$g)$ urgent case: a doctor treats the patient;

$h)$ non-urgent case: a nurse treats the patient;

$i)$ treatment done: a doctor dismisses the patient;

$j)$ the patient needs additional treatment: she takes an appointment for a specialist examination and leaves (✓).

The workflow enforces a SoD constraint between b and i: the doctor who first evaluates the case must be different from the doctor who dismisses the patient, so as to ensure a more thorough examination of the patient's conditions. For readability, the workflow is graphically represented in Figure 4, where we use the letters $\mathsf{P}, \mathsf{D}, \mathsf{N}$ to represent roles $\mathsf{Patient}, \mathsf{Doctor}, \mathsf{Nurse}$ respectively.

VII. Related Work

The paper by Wang *et al.* on the security of delegation in access control systems [23] was one of the main sources of inspiration for the present work. The paper studies how delegation can be abused by colluding users to bypass the intended security policies in a workflow system. Though the security property we consider in the present paper is an adaptation of the security property in [23] to the case of ARBAC administrative actions, there are several notable differences between this work and [23]. First, static analysis is only briefly mentioned in [23] as a possible way to check security, but no static analysis is actually proposed by the authors: rather, the solution they develop requires an extension of the workflow system with additional runtime checks, which is inconvenient or even impossible in many practical scenarios. Second, we experimentally validate the applicability of our theory: developing a sound - yet precise - static analysis for workflow systems built on top of realistic ARBAC policies is hard, since intractability lurks around the corner [20]; indeed, we observed the need to aggressively optimise our encoding into ARBAC to obtain an efficient static analysis. Finally, the formal model in this paper is quite different and more general than the one in [23]: in particular, we focus on the ARBAC standard rather than on an ad-hoc extension of RBAC with delegation and we consider a more expressive model of workflows based on stable event structures rather than on a partially ordered set of tasks.

The satisfiability (or consistency) of workflows is a classic problem in computer security [21], [22]. Roughly, a workflow is satisfiable with respect to a given user-to-role assignment UR if and only if the users included in UR are able to complete it. Checking security against collusion (Definition 9) may require one to check the satisfiability of a workflow. However, as we discussed, checking security against collusion requires one to generalize algorithms for workflow satisfiability to deal with the presence of administrative actions changing the initial user-to-role assignment UR. Building all the possible user-to-role assignments and checking satisfiability with respect to

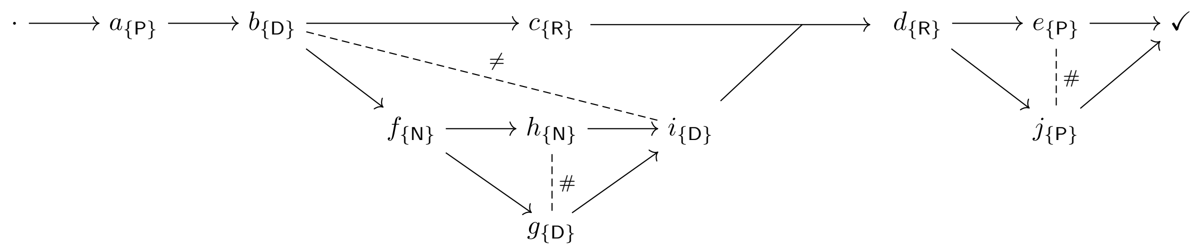

Figure 4. Case study: a first aid procedure

them is not feasible in practice, given the exponential blow-up of the possible role combinations assigned to the users of the system and the fact that workflow satisfiability is NP-hard in most practical cases [22].

Crampton and Khambhammettu proposed algorithms to check the satisfiability of workflows supporting delegation operations [11]. These algorithms ensure that permitting a delegation request does not prevent the completion of a workflow. Their goal is then essentially dual to the static analysis in this paper, which ensures that administrative actions cannot be abused to complete a workflow which could not be completed under the original user-to-role assignment. Indeed, one should observe that there is often a trade-off between security and business continuity: if collusions are the main concern for system administrators, the present paper proposes a viable solution; if instead it is better to ensure workflow termination at the price of permitting collusion, the approach in [11] should be considered. We argue that these considerations strongly depend on the application scenario, the workflow semantics and the considered set of users.

Basin *et al.* conducted a formal study on the tension between security policies and business objectives in workflow systems represented as CSP processes [6]. They formalize a notion of *obstruction*, generalizing the notion of deadlock for systems where access control policies are enforced. Roughly, an obstruction happens when the enforcement of an access control policy prevents a possible execution path in a workflow. The paper presents the design and the implementation of an obstruction-free authorization enforcement mechanism for workflow systems.

VIII. Conclusion

We studied the problem of collusion attacks in ARBAC-based workflow systems, where malicious users may change the user-to-role assignment in the attempt of sidestepping the intended security policies. We formulated a formal definition of security against collusion and we proposed a novel static analysis technique which can be used to prove or disprove security for a large class of ARBAC workflow systems. We discussed how to aggressively optimise the static analysis to ensure its efficiency in practice and we showed the feasibility of our approach by implementing a tool, WARBAC, and by performing an experimental evaluation on a set of publicly available examples.

There are many avenues for future work. We would like to extend our theory to the case of workflows including loops, which are quite popular in practice, but were left out from the present paper for the sake of simplicity, most notably because the interaction between loops and BoD/SoD constraints is quite subtle [6]. Moreover, we plan to design and implement a translator from high-level workflow description languages like BPMN into event structures, thus making WARBAC easier to use. Again on the practical side, we plan to extend WARBAC with a module which, given a role reachability trace returned by VAC, verifies whether this trace actually corresponds to a successful trace of the workflow system: this would be very useful to improve the practicality of WARBAC in absence of a stronger completeness result for our static analysis.

Acknowledgements: We would like to thank the anonymous reviewers for their useful suggestions. This work was partially supported by the MIUR project ADAPT.

References

[1] CBMC tool home page. http://www.cprover.org/cbmc/, accessed: 02-11-2016

[2] INTERPROC tool home page. http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc/index.html, accessed: 02-11-2016

[3] NuSMV tool home page. http://nusmv.fbk.eu/, accessed: 02-11-2016

[4] Ahn, G., Sandhu, R.S., Kang, M.H., Park, J.S.: Injecting RBAC to secure a web-based workflow system. In: ACM Workshop on Role-Based Access Control. pp. 1–10 (2000)

[5] Azghandi, N.G.: Petri nets, probability and event structures. Ph.D. thesis, University of Edinburgh (2014)

[6] Basin, D.A., Burri, S.J., Karjoth, G.: Obstruction-free authorization enforcement: Aligning security and business objectives. Journal of Computer Security 22(5), 661–698 (2014)

[7] Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. ACM Trans. Inf. Syst. Secur. 2(1), 65–104 (1999)

[8] Botha, R.A., Eloff, J.H.P.: Separation of duties for access control enforcement in workflow environments. IBM Systems Journal 40(3), 666–682 (2001)

[9] Calzavara, S., Rabitti, A., Bugliesi, M.: Compositional typed analysis of ARBAC policies. In: IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015. pp. 33–45 (2015)

[10] Calzavara, S., Rabitti, A., Steffinlongo, E., Bugliesi, M.: Static detection of collusion attacks in ARBAC-based workflow systems. Tech. rep. (2016), available at https://sites.google.com/site/warbacanalyser/

[11] Crampton, J., Khambhammettu, H.: Delegation and satisfiability in workflow systems. In: SACMAT 2008, 13th ACM Symposium on Access Control Models and Technologies, Estes Park, CO, USA, June 11-13, 2008, Proceedings. pp. 31–40 (2008)

[12] 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)

[13] Ferrara, A.L., Madhusudan, P., Nguyen, T.L., Parlato, G.: Vac - Verifier of administrative role-based access control policies. In: Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. pp. 184–191 (2014)

[14] Ferrara, A.L., Madhusudan, P., Parlato, G.: Security analysis of role-based access control through program verification. In: 25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012. pp. 113–125 (2012)

[15] Ferrara, A.L., Madhusudan, P., Parlato, G.: Policy analysis for self-administrated role-based access control. In: Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings. pp. 432–447 (2013)

[16] 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)

[17] Jha, S., Li, N., Tripunitara, M.V., Wang, Q., Winsborough, W.H.: Towards formal verification of role-based access control policies. IEEE Trans. Dependable Sec. Comput. 5(4), 242–255 (2008)

[18] Langerak, R., Brinksma, E., Katoen, J.: Causal ambiguity and partial orders in event structures. In: CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings. pp. 317–331 (1997)

[19] Li, N., Tripunitara, M.V., Bizri, Z.: On mutually exclusive roles and separation-of-duty. ACM Trans. Inf. Syst. Secur. 10(2) (2007)

[20] 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)

[21] Tan, K., Crampton, J., Gunter, C.A.: The consistency of task-based authorization constraints in workflow systems. In: 17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA. p. 155 (2004)

[22] Wang, Q., Li, N.: Satisfiability and resiliency in workflow authorization systems. ACM Trans. Inf. Syst. Secur. 13(4), 40 (2010)

[23] Wang, Q., Li, N., Chen, H.: On the security of delegation in access control systems. In: Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings. pp. 317–332 (2008)

[24] Winskel, G.: Event structures. In: Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986. pp. 325–392 (1986)

[25] Winskel, G.: An introduction to event structures. In: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30 - June 3, 1988, Proceedings. pp. 364–397 (1988)