Security Policy Satisfiability and Failure Resilience in Workflows

Meghna Lowalekar, Ritesh Kumar Tiwari, and Kamalakar Karlapalem

Center for Data Engineering,
International Institute of Information Technology,
Hyderabad, India - 500032
meghnal@students.iiit.ac.in, ritesh@research.iiit.ac.in, kamal@iiit.ac.in

Abstract. *Security policy satisfiability* and *high failure resilience* (i.e. survivability) are desirable properties of every system. Security issues and failure resilience are usually treated in stand alone mode and not in synergy. In this paper, we bridge this gap for workflows. We propose techniques which ensure that user-task assignment is both secure and failure resilient and present frameworks that meet different criteria of *security policy*, *security constraints*, and *failure resilience*.

1 Introduction

A user is capable of doing certain tasks in an organizational workflow. But from the security perspective, all the information and resources cannot be made accessible to every user as allowing such uncontrolled access gives unbounded privileges to the user, thereby increasing the chances of attack and subsequent damage. Hence an access control mechanism based on *user capability* that satisfies the *security policy* and *constraints* is needed for assigning users to tasks and their subsequent enactment during runtime. *Failure resilience* (survivability) is a pivotal issue in any organization. Current state of art focuses mainly on fault tolerance at the resource level. It is evident that users can also fail (or be unavailable). Hence there is a need to focus on user level failure resilience for ensuring overall system survivability. For achieving high failure resilience at the user level, a user should have the capability to do a large number of tasks, which results in providing each user with access to a lot of information, thereby increasing the chances of knowledge attacks.

Consider the tendering process which involves many tasks such as advertisement of the requirement for goods or services, preparation of tender documents, registration of suppliers, response to tenders (filling of quotations), evaluation of responses to tenders and finally awarding the contract to a supplier. The process of tender management (including tasks involved and why they are performed) is a company property and should be preserved. In case of large tenders the process of responding to tenders also involves many steps and many users. The response should be submitted by due date even if some users who are involved in response preparation/processing are absent. A user should not get the complete knowledge about the tendering process as this can lead to knowledge attack. Therefore, it is important to achieve security along with failure resilience (tendering process is deadline driven and should be completed even if some users are not

V. Matyáš et al. (Eds.): The Future of Identity, IFIP AICT 298, pp. 197–210, 2009.

present). Similarly in case of defense procedures where tasks are very critical and delay due to absence of any user is not allowed the problem of achieving high failure resilience along with security is important.

1.1 Related Work

Hung et al. [1] present the security features of workflow systems. They discussed the trade off between security and failure resilience. They have proposed a greedy algorithm that determines task assignments that would achieve high failure resilience and low security risk factor. In [1], access control policies and separation of duty constraints are not considered.

Li et al. [2] introduced the concept of resilience policies in access control. Resilient policies ensure that access is properly enabled so that a critical task can be completed even in the absence of some users. Their work mainly focuses on checking the satisfiability of a resilience policy in an access control state. They have shown the complexity of the problem. They also described methods to determine whether a resilient policy is consistent with the separation of duty policies.

Wang et al. [3] studied the resiliency problem in workflow systems. They described that the resiliency in workflow systems differs from resilience policies. In a workflow system, due to the existence of authorization constraints, there is a possibility that even if a set of users together have the permission to perform all steps of the workflow, they can not complete the task. They defined three levels of resiliency: *static*, *decremental* and *dynamic*. The work mainly focuses on checking whether a workflow model is resilient or not i.e whether a workflow can be completed in the absence of some users.

1.2 Contributions and Organization of Paper

In this paper, we focus on operational failure resilience and access control in a user-based system. Ideally, we want a user-task assignment which is both failure resilient and secure (i.e. it satisfies the organizational security policy and associated constraints and also does not provide a user access to a lot of information). To achieve this goal, we use the following two approaches which are described in section 3.

i. Generating *all possible* user-task assignments which satisfy security policy and constraints (refer section 3.1).
ii. Formulating the problem using *Quadratic Programming*(refer section 3.2).

Our work focuses on finding user-task assignments satisfying policy and separation of duty constraints such that the workflow is *min* $\mathcal{K}$ failure resilient i.e workflow can be completed even if $\mathcal{K}$ users fail.

The paper is organized as follows. In section 2, we describe the preliminaries and calculation of failure resilience. In section 3, we explain the approaches for computation of user-task assignments. In sections 4 we present results and in section 5, we conclude with future work.

2 Preliminaries

A workflow can be defined as a set of tasks ($\mathbb{T}$) coordinated by a set of events ($\mathbb{E}$) whose successful execution results in the completion of an instance of the activity. The sequence of tasks in a workflow can have:

i. *Sequential constraints* ($T_i \prec T_j$): Execution of task T_i should be completed before T_j starts executing.
ii. *Temporal constraints*: Temporal constraints can be further classified as *activation time constraints* and *execution time constraints*. Activation time constraints (i.e. $\{T_i\}^{activated_{[p,\ q)}}$) denotes that task T_i can be activated by an authorized user only within time period $[p,q)$. Execution time constraints (i.e. $\{T_i\}^{exec_p}$) denote that task T_i can be executed by an authorized user for atmost p time units after the invocation.

Workflow tasks can be treated as a combination of automated and manual processes which are represented (and controlled) by users. In most of the practical workflows, user-task assignment is static due to specialized users that can do specific tasks or due to initial work assignment to users. The approaches proposed in this paper, to achieve failure resilience with access control are applicable for scenarios where Task-Based Authorization Control [4] is used for enforcing access control on users.

Table 1. Notations Used

$U_1, U_2, \ldots, U_n$	: Users
$T_1, T_2, \ldots, T_n$	: Tasks
pol	: Policy
$C_1, C_2, \ldots, C_n$	: Constraints
cap	: Capability
FR_{T_i}	: Failure resilience of task T_i
$FR_{activity}$	: Failure resilience of activity
$exec(T_i)$	: Set of users having capability to execute task T_i
$assigned(T_i)$	: Set of users assigned to task T_i

2.1 User-Task Assignment

Let *capability of a user* (cap: $U \xrightarrow{cap} \{T\}$) denote the set of tasks that the user is capable of doing. Similarly, *executability of a task* $\left[exec(T_k)\right]$ denotes the set of users that possess the capability to execute *task* T_k. Using the capability sets of users, we can compute the executability set corresponding to every task of the workflow.

A user might possess the capabilities to perform all tasks of a workflow but the organization security policy can prevent it from performing some. Thus, if a user possesses the capability of performing a task, it does not necessarily imply that it is assigned to the task. But if a user does not possess the capability, then it *can not* be assigned to the task. The organization security policy forces restrictions on the capability sets of users and hence the executability set of a task. Separation of duty constraints further reduce this set. If $\left[assigned(T_k)\right]$ denotes the set of users that are assigned to task T_k after satisfying security policy and constraints, then $assigned(T_k) \subseteq exec(T_k)$. Out of all the users that are assigned to tasks, one user per task is chosen for executing the task.

Security policy defines which users are authorized to execute tasks based on organization security requirements. The term *policy* used in this paper encapsulates the notion of both *confidentiality* and *integrity policy* associated with access control. *Security constraints* place additional (*activation* and/or *privilege level*) restrictions on users and tasks that satisfy the security policy.

Security constraints enforced on users can be classified [5] into:

i. *Temporal* constraints,
ii. *Separation of duty* (*Static*[1]*/ Dynamic/ Operational*[2]*/ Object Based*) constraints, and
iii. *Location* constraints.

There are two types of scenarios considered in the paper, which are as follows:

1. *Purely static* - All user task assignments are fixed before the execution of workflow. Users are assigned to tasks considering all the constraints and this assignment does not change at runtime. Failure resilience has a fixed value.
2. *Purely dynamic*- Before the execution of workflow users are assigned to tasks considering all the constraints but assignment can change at runtime to get more failure resilience. Failure resilience changes dynamically but has a lower bound.

In static scenario, for all the tasks of a workflow, the set $assigned(T_k)$ can be computed using any of the approaches described in section 3. In case of dynamic scenario quadratic programming approach (section3.2) can be used. Section 3.2 also shows how to apply quadratic programming to change the assignments at runtime.

In the next part, we show the computation of failure resilience for a workflow activity. The definitions and formulae hold for purely static scenario. For purely dynamic scenario these constitutes the lower bound of failure resilience. This is because, in dynamic scenario assignment is changed to get more failure resilience, therefore, we will get the failure resilience which we were getting in static case.

2.2 Failure Resilience of Task and Activity

Failure resilience of a task denotes the maximum number of user failures a task can handle. Similarly, failure resilience of an activity is the maximum number of user failures in the presence of which activity execution can continue uninterrupted. The activity will fail when any of its constituent task can not be completed successfully. Therefore, failure resilience of activity depends on the failure resilience of its constituent tasks.

Definition 1. *Failure Resilience of a task is one less than the number of users that are assigned to the task* $\big($*i.e.* $\mathrm{FR}_{T_i} = (|assigned(T_i)| - 1)\big)$.

Lemma 1. *Given an assignment of n users to t tasks, the workflow activity is guaranteed to execute as long as the number of failed users* $\leq min_{\forall_i}(FR_{T_i})$.

[1] If two tasks T_1 and T_2 are in SSoD and if $(U_i, T_1) \in$ *user task assignment* $\Rightarrow (U_i, T_2) \notin$ *user task assignment*.

[2] If $\{T_1, T_2, \ldots T_n\}$ is set of critical tasks in a workflow, then as per operational SoD, any user U_i cannot execute all critical tasks in any instance of workflow.

Proof. For the successful execution of an activity, all its constituent tasks should be completed successfully. The activity will fail if all users in any of the constituent tasks of the activity fail. In the *worst case*, the activity will fail when the number of users that fail is $\lceil min_{\forall i}(|assigned(T_i)|)\rceil$. Therefore, an activity is guaranteed a successful completion when the number of users that fail is $\leq \lceil min_{\forall i}(|assigned(T_i)|) - 1\rceil$.

Definition 2. *Failure resilience of an activity is the minimum of failure resilience of its constituent tasks i.e* $\mathrm{FR}_{activity} = min_{\forall i}(FR_{T_i})$

Corollary 1. *Given an assignment of n users to t tasks and the number of failed users* $> min_{\forall_i}(FR_{T_i})$*, the activity can still continue; but there exists at least one specific combination of* $min_{\forall_i}(FR_{T_i})$ *users whose failure will fail both a task and the activity.*

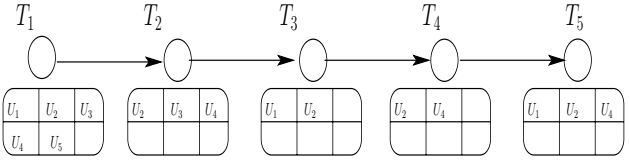

Fig. 1. Workflow activity consisting of five tasks

Example 1. Figure 1 shows five tasks and users assigned to each of them. As tasks T_3 and T_4 have the minimum number of users, in the worst case, an activity will fail when either of the user sets (U_1, U_2) or (U_2, U_4) fails. However, if user set (U_1, U_3, U_4) fails, even then the activity will be successfully completed. Therefore, it will not always be the case that when $\lceil min_{\forall i}(|assigned(T_i)|)\rceil$ users fail, then activity fails too.

Note that we have not considered sequence constraints in doing failure resilient user-task assignment. Failure resilience is independent of simple task precedence. Constrained precedence can be handled by incorporating them as SoD constraints (refer Proposition 1).

Relationship between Task Precedence and Failure Resilience. Consider a simplistic workflow consisting of three tasks T_1, T_2, T_3. Let T_1 and T_2 have a precedence relationship ($T_1 \prec T_2$) while T_3 does not have any precedence relationship ($(T_1, T_2) \not\prec T_3$). Precedence relationships between the tasks can be of two types:

i *Simple precedence*: ($T_1 \prec T_2$) implies that the execution of T_1 precedes T_2, but users in the set $\{assigned(T_1)\}$ will have no dependency relationships with those in $\{assigned(T_2)\}$ at runtime.

ii *Constrained precedence*: ($T_1 \prec^{\mathbb{C}} T_2$) implies that the execution of T_1 precedes T_2 and if $\mathbb{C}$ is a separation of duty constraint, then all users in the set $\{exec(T_1)\}$ will have a dependency relationship with those in $\{exec(T_2)\}$. For example, if user $U_1 \in \{exec(T_1), exec(T_2)\}$ and if it executes T_1, then it cannot execute T_2 in that workflow instance. But as the set $\forall k\{assigned(T_k)\}$ is calculated after taking all security constraints into account (refer section 3), users in $\{assigned(T_1)\}$ will have no dependency relationship with users in $\{assigned(T_2)\}$.

Proposition 1. ***Precedence relationships do not have any implication on failure resilience.***

Reason. Consider the above example. As the set $\{assigned(T_1)\}$ has no dependency relationship with $\{assigned(T_2)\}$ and $\{assigned(T_3)\}$, the number of users who can execute T_i (i.e. $|assigned(T_i)|$) in any workflow instance will not depend on task precedence. Hence, task precedence is not a determinant of the failure resilience of the workflow activity.

In this section, we have computed failure resilience for workflows assuming we know the user-task assignment. In the next section, we show how to assign tasks to users to achieve failure resilience.

3 Failure Resilient User-Task Assignments

3.1 Exhaustive Search

In this approach, we achieve our goal in the following manner:

i. Based on user capability, we derive/identify all the tasks an user can possibly perform (without considering the security policy and constraints).
ii. Applying *security policy* restrictions and *separation of duty* constraints, we identify the combinations of user-task assignments that are not allowed in a *secure* state of the system.

    The permitted user task assignments satisfying security policy and constraints can be derived as:

    $\{User\ Task\ Assignment\}_{Step_i} \backslash \{User\ Task\ Assignment\}_{Step_{ii}} =$
    $\{Permitted\ user\ task\ assignment\}$
iii. From the *permitted user task assignment* set, we select the user-task assignments that have *min* $\mathcal{K}$ failure resilience (definition 3).

The diagrammatic representation of the system model for this approach is shown in Figure 2. (*a*) represents the possible user-task assignments considering only the capabilities of users. The assignments that violate the security policy are removed from (*a*) to obtain (*b*). (*c*) gives the different possible combinations of user-task assignments derived from (*b*) that satisfy security constraints. (*d*) is derived from (*c*) to obtain the desired level of failure resilience. The steps that the model follows (Figure 2) to achieve security and failure resilience are described below.

Step (a): Initially, all users are assigned to tasks that they are capable of doing. Therefore, the outcome of executing step *(a)* in Figure 2 will be the set $\left[\ \forall k\ \{exec(T_k)\}\right]$.

Step (b): The executability set of a task after applying the security policy (*pol*) will be a subset of the executability set before applying the security policy. Therefore, $\left[\forall_k \left(\text{exec}(T_k)^{pol} \subseteq \text{exec}(T_k)\right)\right]$ is what we get after executing step *(b)* of Figure 2.

Step (c): Security constraints on users are specified as *rules* in the *rule base* (refer Figure 3). $(T_1)_{U_i} \Rightarrow \neg(T_2)_{U_i}$ represents static SSOD of user U_i over tasks T_1 *and* T_2.

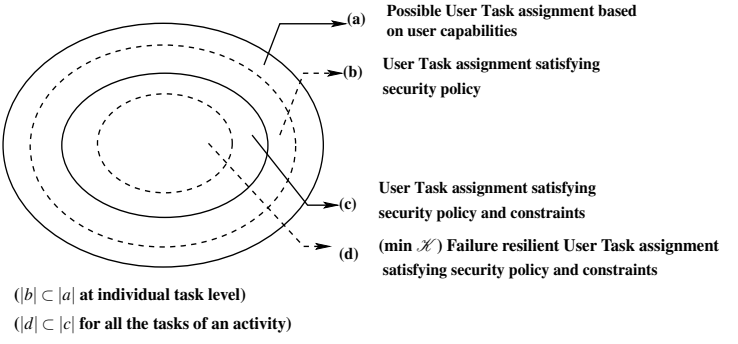

Fig. 2. Stepwise user task assignment

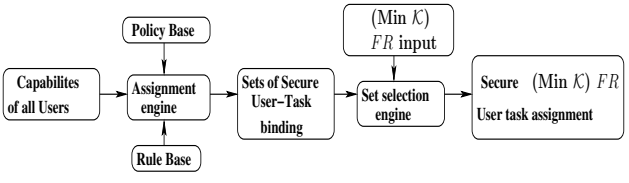

Fig. 3. Conceptual model of proposed system

If there are n security constraints $\mathbb{C} = \{C_1, C_2, \ldots, C_n\}$, then all of them (i.e. $C_1 \wedge C_2 \wedge \ldots \wedge C_n$) need to be satisfied for secure user-task binding. The literals in $C_1, C_2 \ldots$ are all of the form $(T_i)_{U_j}$. We convert $\mathbb{C}$ into *Conjunctive Normal Form* (CNF) and then find all the solutions that satisfy $\mathbb{C}$ using the *modified DPLL algorithm* [6]. Each solution is represented in the form $S_i = \forall_k (U')^i_{T_k}$ where S_i is the i^{th} solution and $(U')^i_{T_k}$ represents the set of users which *cannot perform* task T_k in the i^{th} solution. We need user-task bindings which satisfy both security policy and constraints. Therefore, $(U')^i_{T_k}$ should be removed from $exec(T_k)^{pol}$. For the i^{th} solution, $\left[(exec(T_k)^{pol+\mathbb{C}})^i = exec(T_k)^{pol} - (U')^i_{T_k}\right]$ is the set of users which can be assigned to task T_k. The outcome of step 6 of *Algorithm1* is the solution for step (c) of Figure 2. The time taken by this approach largely depends on this step.

Step (d): From the previous step, we have all possible sets of user-task assignment that satisfy both the security policy and constraints. All these sets give a different secure user-task assignment. But it is desirable to have failure resilience along with security. Here we introduce the notion of *min* $\mathcal{K}$ failure resilience.

Definition 3. *min* $\mathcal{K}$ *failure resilience: For achieving min* $\mathcal{K}$ *FR for an activity, there should exist a set of user-task assignments in the activity for which* $FR_{activity} \geq \mathcal{K}$.

If $\mathcal{K}$ is greater than maximum achievable failure resilience then solution can not be computed.

Table 2. Algorithm for *constraint satisfiable* User-Task assignment

Algorithm1: Constraint Satisfiable User-Task Assignment($\mathbb{C}$)	
1.	**Input:** [i.] $\forall_k(exec(T_k)^{pol})$
	[ii.] Constraints $\mathbb{C}$
2.	$\forall_i(C_i \in \mathbb{C})$, generate CNF of literals for C_i
3.	Generate $\wedge_{(\forall i)}(C_i)$
4.	Generate all solutions (sets of User-Task assignments) which satisfy $\wedge_{(\forall i)}(C_i)$ using *modified DPLL* algorithm [6].
5.	Convert all generated solutions ($\forall_i(S_i)$) in form $\forall_i\left[S_i = \forall_k(U')^i_{T_k}\right]$ where $(U')^i_{T_k}$ = Set of users that cannot do T_k in i^{th} solution.
6.	$\forall_i, \forall_k\left[(exec(T_k)^{pol+\mathbb{C}})^i = exec(T_k)^{pol} - (U')^i_{T_k}\right]$

Definition 4. *Maximum achievable failure resilience: Maximum achievable failure resilience of an activity is the maximum value of failure resilience which can be achieved for the activity satisfying security policy and separation of duty constraints.*

If $\mathcal{K} <$ *Maximum achievable failure resilience* and for the i^{th} solution ($FR^i_{activity} \geq \mathcal{K}$), then $\forall_k(exec(T_k)^{pol+\mathbb{C}})^i$ will give the *min* $\mathcal{K}$ failure resilient user task assignment corresponding to that solution. There can be more than one solution which have $FR_{activity} \geq \mathcal{K}$; in that case first we look for the solution with minimum average number of tasks per user. If the average number of task per user is same for two solutions, then the solution with minimum variance of number of tasks an user is doing is chosen (as it minimizes the knowledge gained by users). If the chosen solution is the p^{th} solution then $\forall_k(exec(T_k)^{pol+\mathbb{C}})^p$ is the set which constitutes step (d) of Figure 2 and also the set $\forall_k$ *assigned*(T_k), that is,

$$\forall_k \; assigned(T_k) = \forall_k(exec(T_k)^{pol+\mathbb{C}})^p$$

3.2 Quadratic Programming Approach

The problem of achieving *min*$\mathcal{K}$ failure resilience while satisfying the security policy and constraints is formulated in the form of a 0-1 quadratic programming problem. Let $T = \{T_1, T_2, ...T_n\}$ be a set of n tasks and $U = \{U_1, U_2, ..., U_m\}$ be a set of m users which will be assigned to the tasks. $X_{ij}(i = 1,2,...,n; j = 1,2,...,m)$ is used to denote assignment of user j to task i. X_{ij} is 1 if user j is assigned to task i, and is 0 otherwise.

	U_1	U_2	...	U_m
T_1	X_{11}	X_{12}	...	X_{1m}
T_2	X_{21}	X_{22}	...	X_{2m}
...	...	...	...	...
T_n	X_{n1}	X_{n2}	...	X_{nm}

We need to assign users to tasks by taking into account the *capabilities* of users, *security policy* and *Separation of duty constraints*. As described in section 2, if a user does

not possess the capability to perform a task, then it can not be assigned to that task. Therefore, X_{ij} is set to 0 if U_j is not capable of performing task T_i i.e. $T_i \notin cap(U_j)$. If U_j can not be assigned to T_i as per the security policy, then also X_{ij} is set to 0. We want to achieve min $\mathcal{K}$ failure resilience while satisfying separation of duty constraints. The separation of duty constraints are expressed in the form of inequality constraints.

If T_1, T_2 are in static SoD then T_1 *and* T_2 both can not be assigned to user U_i simultaneously. Therefore, X_{1i} and X_{2i} both can not be 1 at the same time i.e.

$$X_{1i} + X_{2i} \leq 1 \tag{1}$$

Similarly, if a set of p tasks $T_1, T_2, ..., T_p$ is in static SoD, then no two of them can be assigned simultaneously to user U_i, that is,

$$X_{1i} + X_{2i} + ... + X_{pi} \leq 1$$

For operational SoD, if T_1, T_2 are critical tasks for user U_i, then U_i can not execute both T_1 *and* T_2 at runtime, hence X_{1i} and X_{2i} can not be 1 at the same time, that is,

$$X_{1i} + X_{2i} \leq 1 \tag{2}$$

Similarly, if there is a set of q critical tasks $T_1, T_2, ..., T_q$, all of which can not be done by user U_i simultaneously then at least one of $(X_{ji}) < j = 1, 2, .., u >$ should be 0, that is,

$$X_{1i} + X_{2i} + ... + X_{qi} \leq q - 1$$

To achieve min $\mathcal{K}$ failure resilience each task should be assigned to at least $\mathcal{K}$ users. Therefore,

$$\forall_i \sum_{j=0}^{m} X_{ij} \geq \mathcal{K} + 1 \tag{3}$$

Knowledge gained by a user by executing the workflow is the weighted sum of knowledge gained in doing its constituent tasks. Let w_i $1 \leq i \leq n$ denote the weight corresponding to each task (T_i $1 \leq i \leq n$) of a workflow then total knowledge gained by user U_j is:

$$\sum_{i=1}^{n} w_i * X_{ij}$$

If v_j $1 \leq j \leq m$ denote the weight of a user (U_j $1 \leq j \leq m$) (a user with higher weight should be more knowledgeable then user with lower weight). Therefore, in order to minimize the risk, any user should not have more knowledge in proportion to its weight. Therefore we should minimize the weighted standard deviation and hence weighted variance. Weighted average of knowledge gained by user

$$weightedavg = (\sum_{j=1}^{m} v_j * \sum_{i=1}^{n} w_i * X_{ij}) / \sum_{j=1}^{m} v_j$$

Variance, which is the objective function of *QPP* and is to be *minimized*, is given by

$$variance = (\sum_{j=1}^{m} v_j * \left[(\sum_{i=1}^{n} X_{ij}) - weightedavg \right]^2) / \sum_{j=1}^{m} v_j \tag{4}$$

The weights are subjective in nature and there is no known scientific standard which can measure the knowledge of a task of an activity and the knowledge gained a user by performing the activity. For simplicity we have assumed, all tasks and users are of equal weight and hence $\forall_{i=1,2,..,n}\ w_i = 1$ *and* $\forall_{j=1,2..,m}\ v_j = 1$

Therefore, this is a 0-1 quadratic programming problem(*QPP*) with linear constraints ((1, 2 and 3) and convex quadratic objective function(4), which can be solved using any of the available MIQP (mixed integer quadratic programming) solver.
After computing the values of $\forall_{i,j} X_{ij}$,if X_{ij}=1 then, user U_j is assigned to task T_i i.e, $assigned(T_i) = \forall j|_{X_{ij}\neq 0} U_j$. If $\mathcal{K} > Maximum\ achievable\ failure\ resilience$ then solution with Maximum Achievable failure resilience is obtained.(refer Algorithm 2).

The users in the set $assigned(T_k)$ will be given privileges to perform the task T_k. The assignment of privileges to users is either *static* (at activation time) or *dynamic* (at runtime). In case of static assignment of privileges, all privileges are given to the users at the very beginning and are retained forever. In case of dynamic assignment of privileges, privileges are given just before the task is to be executed. The privileges are revoked after the task has executed.

Table 3. Algorithm for *user-task assignment* using QPP

Algorithm2: User-Task assignment with QPP	
1.	$\mathcal{K}$ is expected failure resilience.
2.	Form QPP with constraints 1 or 2 and 3 with objective function(4)
3.	Solve QPP with MIQP solver
4.	while *solution* is not *feasible*
5.	modify constraint 3 replace $\mathcal{K}$ by $\mathcal{K}-1 (\mathcal{K} = \mathcal{K}-1)$
6.	Solve modified QPP with MIQP solver
7.	Solution is obtained with Failure resilience $\mathcal{K}$.

Change of Assignment in Dynamic Scenario. In case of dynamic scenario we get the initial user-task assignment by solving the *QPP*. On failure of a user, this assignment can be modified at runtime to get more failure resilience. Consider the workflow in figure fig4,

There are four tasks T_1, T_2, T_3 *and* T_4. Suppose there are five users U_1, U_2, U_3, U_4 *and* U_5. All users can do all the tasks of the workflow as per security policy. Let us say $(T_1, T_2), (T_2, T_3)$ *and* (T_3, T_4) are sets of mutually exclusive tasks, so no user can do two tasks in a set simultaneously. Also, let us assume that we need a failure resilience of 2.

In this case, the initial *QPP* is:

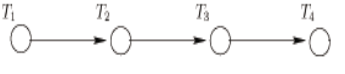

Fig. 4. Example workflow

minimize

$$variance = (\sum_{j=1}^{5} [\sum_{i=1}^{4} X_{ij} - avg]^2)/5$$

where

$$avg = (\sum_{j=1}^{5}\sum_{i=1}^{4} X_{ij})/5$$

subject to constraints

SoD constraints

$X_{11}+X_{21} \leq 1; X_{12}+X_{22} \leq 1; X_{13}+X_{23} \leq 1; X_{14}+X_{24} \leq 1; X_{15}+X_{25} \leq 1$
$X_{21}+X_{31} \leq 1; X_{22}+X_{32} \leq 1; X_{23}+X_{33} \leq 1; X_{24}+X_{34} \leq 1; X_{25}+X_{35} \leq 1$
$X_{31}+X_{41} \leq 1; X_{32}+X_{42} \leq 1; X_{33}+X_{43} \leq 1; X_{34}+X_{44} \leq 1; X_{35}+X_{45} \leq 1$

Failure resilience constraints

$X_{11}+X_{12}+X_{13}+X_{14}+X_{15} \geq 2; X_{21}+X_{22}+X_{23}+X_{24}+X_{25} \geq 2$
$X_{31}+X_{32}+X_{33}+X_{34}+X_{35} \geq 2; X_{41}+X_{42}+X_{43}+X_{44}+X_{45} \geq 2$

The solution to this *QPP* is:

$assigned(T_1) = (U_5, U_3)$;$assigned(T_2) = (U_1, U_4)$;$assigned(T_3) = (U_3, U_2)$
$assigned(T_4) = (U_4, U_1)$

This is the user-task assignment in case we take static user-task assignment and is the initial assignment for the dynamic case. Now suppose user U_5 has executed task T_1 and user U_1 has executed task T_2. If user U_3 fails at this stage, then in static scenario user U_2 executes T_3, assignment remains fixed and failure resilience at this stage is 1. On the other hand, in case of dynamic scenario, a new *QPP* is formed as given below. The objective function remains the same as we still want to minimize the knowledge gain. The constraints which were there still hold on as tasks are still mutually exclusive. There will be some new constraints which are as follows:

$\forall_i X_{i3} = 0$ % As user U_3 failed
$X_{15} = 1; X_{21} = 1$ % As U_5 *and* U_1 executed T_1 *and* T_2 respectively, so this assignment can not be changed.

On solving the new *QPP* we get the assignment as:

$assigned(T_1) = (U_5, U_4)$;$assigned(T_2) = (U_1, U_2)$;$assigned(T_3) = (U_4, U_5)$
$assigned(T_4) = (U_1, U_2)$

Now U_4 executes T_3 and we can get a failure resilience of 2 even after the failure of a user.

As shown in the example we can get a high failure resilience by changing the assignment at runtime. To change the assignment at runtime, we use iterative quadratic programming approach. We form a new *QPP* at each failure by introducing the constraints arose due to failure of a user and also because of execution of tasks preceding the task at which failure occurs(shown in the example).

The problem with the iterative quadratic programming approach is that the time taken to solve *QPP* at runtime increases the time of execution of workflow. But, since the number of unknown variables reduces in each iteration, it takes less time to solve resulting *QPP*.

4 Results

We carried out our experiments on a 2.8 GHz processor with 512 MB of RAM. The two approaches described in the paper are both NP hard but there are commercial tools available for solving quadratic programming problems which can compute the solution really fast. For solving MIQP, we used ILOG OPL-CPLEX Analyst Studio [7] which provide the fastest possible execution times.

Test cases [8] are randomly generated. Table 5 shows the time taken by exhaustive search approach and the quadratic programming approach. In *time* column for exhaustive search approach, if the solution can not be generated within 1000 seconds then a '-' is kept. In *time* column for quadratic programming approach the value *before/* indicates the time taken to compute a feasible solution and value */after* indicates the time taken to compute the optimal solution. If optimal solution is not computed within 60 seconds then a '-' is put. The ILOG CPLEX [9] finds a good feasible solution early but it takes time to prove that solution is optimal. A good feasible solution is one which satisfies all the constraints and the value of objective function for this solution is very close[3] to optimal value (minimum value of variance).

Table 4. Results: Failure resilience for some examples [8]

Instance Name	*No. of users*	*No. of tasks*	*Total Const-raints*[4]	SoD *Const-raints*	max achievable FR	Avg no. of tasks per user	min no.of tasks assigned to a user	max.no of tasks assigned to a user
50_5_2_1.*mod*	5	50	332	240	1	20	19	21
10_10_5_1.*mod*	10	10	61	35	4	5	5	5
15_10_5_1.*mod*	10	15	103	63	4	7.5	7	8
20_15_6_2.*mod*	15	20	239	162	5	8	8	8
30_15_6_3.*mod*	15	30	407	292	5	12	12	12
20_20_9_1.*mod*	20	20	340	238	8	9	9	9
10_30_14_1.*mod*	30	10	259	188	13	4.67	3	5
2_40_22_3.*mod*	40	2	60	46	21	1.1	0	2
10_40_17_1.*mod*	40	10	416	323	16	4.25	4	5
2_50_22_2.*mod*	50	2	125	106	21	0.88	0	2
5_50_22_5.*mod*	50	5	261	207	21	2.2	1	3

Table 4 shows the maximum achievable failure resilience for some of the instances available at [8]. The maximum achievable failure resilience of an activity does not depend on number of users, tasks and constraints. For same number of users, tasks and

[3] The difference between the optimal value and the good value is less than 1%. As all the security constraints are satisfied and *min* $\mathcal{K}$ failure resilience constraint is also satisfied, the solution is secure and failure resilient. The 1% difference affects the knowledge gained by each user. The optimal solution can always be computed but time required will be more. Thus, there is a trade off between the knowledge gained by the users and time taken to compute the assignment.

[4] In case of quadratic programming approach total constraints include failure resilience constraints in addition to SoD constraints. Also the X_{ij} values which need to be initialized to 0 are initialized using the inequality $X_{ij} \leq 0$ (All X_{ij} are integers and are either 0 or 1).

Table 5. Results: Comparison of time taken by the two approaches

No. of users	*No. of tasks*	*Total Constraints*	SoD Constraints	Time(in sec) (Quadratic Programming)	Time (in sec) (Exhaustive Search)
2	2	2	0	1.11/1.11	1
2	2	3	0	0.84/0.84	1
5	5	8	1	0.75/1.53	1.2
5	5	11	2	0.8/1.11	1.4
10	10	54	21	0.51/–	639
10	10	57	26	0.53/–	850
20	20	314	215	2.9/–	–
20	20	329	228	4.0/–	–

constraints maximum achievable failure resilience will be different as it depends on policy and type of constraints. Table 4 also contains average number of tasks, minimum number of tasks and maximum number of tasks assigned to a user. The results (in table 5) show that a feasible solution can always be computed in a small duration using quadratic programming approach. However if more than one feasible solution is generated(whenever a better solution, i.e close to optimal, is generated), then the time for that new solution is recorded. Therefore, the time in the results is the time taken to generate the optimal solution (if optimal solution is not possible within the maximum time limits(60 sec) then closest to optimal solution is considered).

It is evident from the results that formulating the problem as a quadratic programming problem is a better approach as it gives solutions quickly and also does not generate redundant solutions. As time taken is less, the approach is practically applicable in dynamic environments.

5 Conclusion

Many of day to day activities are modeled using workflows. A workflow is a set of tasks which can be executed by a set of users. The users which can execute many of the sensitive and critical tasks of an activity/workflow can be software or humans. Failure to accomplish these critical tasks may lead to delay in activity execution and potential loss to the organization. On the other hand, allowing users to execute multiple critical tasks will lead to potential security attacks through these users (insider attacks). There needs to be a balance between failure resilience constraints and user-task assignments. We have developed two approaches namely *Exhaustive search* and *Quadratic Programming approach* for assigning users to tasks. We have shown that quadratic programming approach is not only efficient but also gives quality results.

The main focus of this work is to provide failure resilience while satisfying security policy and constraints. These concerns are addressed in this paper. We have considered *static* and *decremental* resilience in this paper but plan to incorporate *dynamic* resilience [3] in future. We have considered task based access control, therefore, we are finding resilient user-task assignment. We are working on extending the current framework for role based access control environments [10] where failure resilient user-role assignment need to be identified.

References

1. Hung, P.C.K., Karlapalem, K., GrayIII, J.W.: A Study of Least Privilege in CapBasED-AMS. In: International Conference on Cooperative Information Systems, pp. 208–217 (1998)
2. Li, N., Tripunitara, M.V., Wang, Q.: Resiliency policies in access control. In: ACM Conference on Computer and Communications Security, pp. 113–123 (2006)
3. Wang, Q., Li, N.: Satisfiability and resiliency in workflow systems. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 90–105. Springer, Heidelberg (2007)
4. Thomas, R.K., Sandhu, R.S.: Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Authorization Management. In: Eleventh International Conference on Database Security, pp. 166–181 (1997)
5. Solworth, J.A.: Approvability. In: ASIACCS 2006: ACM Symposium on Information, computer and communications security, pp. 231–242 (2006)
6. Jin, H., Han, H., Somenzi, F.: Efficient Conflict Analysis for Finding All Satisfying Assignments of a Boolean Circuit. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 287–300. Springer, Heidelberg (2005)
7. http://www.ilog.com/products/oplstudio/
8. http://students.iiit.ac.in/~meghnal/inputs/
9. http://eaton.math.rpi.edu/cplex90html/pdf/usrcplex.pdf
10. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-Based Access Control Models. IEEE Computer 29(2), 38–47 (1996)
11. Tan, K., Crampton, J., Gunter, C.A.: The consistency of task-based authorization constraints in workflow systems. In: CSFW, p. 155 (2004)
12. Helsinger, A., Kleinmann, K., Brinn, M.: Framework to Control Emergent Survivability of Multi Agent Systems. In: AAMAS, pp. 28–35 (2004)
13. Navarro, G., Borrell, J., Ortega-Ruiz, J.A., Robles, S.: Access control with safe role assignment for mobile agents. In: AAMAS, pp. 1235–1236 (2005)
14. Kern, A., Walhorn, C.: Rule support for role-based access control. In: ACM symposium on Access control models and technologies, pp. 130–138 (2005)
15. Crampton, J.: A reference monitor for workflow systems with constrained task execution. In: SACMAT, pp. 38–47 (2005)