Resiliency Policies in Access Control Revisited

Jason Crampton
Royal Holloway
University of London
Egham, TW20 9QY
United Kingdom
jason.crampton@rhul.ac.uk

Gregory Gutin
Royal Holloway
University of London
Egham, TW20 9QY
United Kingdom
gutin@cs.rhul.ac.uk

Rémi Watrigant
Royal Holloway
University of London
Egham, TW20 9QY
United Kingdom
remi.watrigant@rhul.ac.uk

ABSTRACT

Resiliency is a relatively new topic in the context of access control. Informally, it refers to the extent to which a multi-user computer system, subject to an authorization policy, is able to continue functioning if a number of authorized users are unavailable. Several interesting problems connected to resiliency were introduced by Li, Wang and Tripunitara [13], many of which were found to be intractable. In this paper, we show that these resiliency problems have unexpected connections with the workflow satisfiability problem (WSP). In particular, we show that an instance of the resiliency checking problem (RCP) may be reduced to an instance of WSP. We then demonstrate that recent advances in our understanding of WSP enable us to develop fixed-parameter tractable algorithms for RCP. Moreover, these algorithms are likely to be useful in practice, given recent experimental work demonstrating the advantages of bespoke algorithms to solve WSP. We also generalize RCP in several different ways, showing in each case how to adapt the reduction to WSP. Li *et al.* also showed that the coexistence of resiliency policies and static separation-of-duty policies gives rise to further interesting questions. We show how our reduction of RCP to WSP may be extended to solve these problems as well and establish that they are also fixed-parameter tractable.

CCS Concepts

•Security and privacy → Access control; *Security requirements;* **•Theory of computation → Fixed parameter tractability;**

Keywords

access control, resiliency, workflow satisfiability, computational complexity, fixed-parameter tractability

SACMAT'16, June 05-08, 2016, Shanghai, China
© 2016 ACM. ISBN 978-1-4503-3802-8/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2914642.2914650

1. INTRODUCTION

Access control is a fundamental aspect of the security of any multi-user computing system, and is typically based on the idea of specifying and enforcing an authorization policy. Such a policy identifies which interactions between users and resources are to be allowed by the system.

Over the last twenty years, authorization policies have become more complex, not least because of the introduction of constraints, which further refine an authorization policy. A separation-of-duty constraint (also known as the "two-man rule" or "four-eyes policy") may, for example, require that no single user is authorized for some particularly sensitive group of resources. Such a constraint is typically used to prevent misuse of the system by a single user.

In the context of workflow systems, such constraints may mean that it is impossible to complete all the steps in a workflow, while simultaneously ensuring that each step is executed by an authorized user and all constraints are satisfied. Hence, there has been considerable interest in analyzing workflow specifications to determine whether they are "satisfiable" or not [3, 4, 9, 11, 15]. Here, the intent of the analysis is to confirm that a system satisfies functional or operational requirements, in the presence of security policies and constraints.

More recently, we have seen the introduction of *resiliency policies*, whose satisfaction indicates a system will continue to function as intended in the absence of some number of authorized users [13, 15]. The purpose of specifying such policies is again to determine whether or not a system satisfies certain operational requirements. Li, Wang and Tripunitara's seminal work on resiliency in access control [13] introduces a number of problems associated with the simultaneous satisfaction of separation-of-duty constraints and resiliency policies. They established that many of these problems were intractable, although certain special cases did admit polynomial-time algorithms. It is these problems that are the focus of this paper.

One interesting aspect of the work by Li *et al.* is the relative sizes of the parameters. In particular, it may be assumed for practical purposes that certain parameters are significantly smaller than others. With this in mind, we exploit the theory of fixed-parameter tractability to investigate the problems introduced by Li *et al.* [13]. Informally, we develop algorithms whose running time is polynomial in the large parameters but may be exponential in the small parameters. Such algorithms can be very useful, particularly if the assumptions underlying the relative sizes of the parameters hold in practice.

The fundamental contribution of this paper is to exhibit a polynomial-time construction to transform an instance of the resiliency checking problem (RCP) to an instance of the workflow satisfiability problem (WSP). Recent advances in the understanding of WSP enable us to characterize the complexity of RCP, in order to obtain efficient algorithms. Moreover, we are able to exploit particular characteristics of the WSP instances obtained from RCP to specialize existing algorithms for solving WSP. In particular:

- we establish that RCP is fixed-parameter tractable for parameters which are likely to take small values in practice;
- we adapt the *Pattern Backtracking* algorithm for WSP [3] to solve transformed instances of RCP;
- we generalize RCP to model additional situations that may be of practical interest, and establish that the reduction to WSP also holds in these cases.

We then consider several problems arising from the interaction of resiliency policies and static separation-of-duty policies. We use a similar reduction to the workflow satisfiability problem to establish that these problems are also fixed-parameter tractable.

In the next section, we introduce relevant concepts from the literature, including resiliency policies, WSP and fixed-parameter tractability. In Section 3, we show how to reduce RCP to WSP, and adapt a known algorithm in order to obtain a better running time. We also consider several generalizations of the problem which are likely to be of practical interest. In Section 4, we extend our approach to find a fixed-parameter tractable algorithm for the policy consistency checking problem. In doing so, we introduce a simpler variant of this problem, called the *mixed policy checking problem*, that is of independent interest. We conclude the paper with a summary of our contributions and some suggestions for future work.

Due to the page limits, the proofs of Theorem 2 and Lemma 2 are given in the appendix.

2. BACKGROUND

In this section, we recall relevant material on resiliency policies, workflow satisfiability and fixed-parameter tractability. We assume there exists a set of resources R, access to which must be controlled, by specifying and enforcing authorization policies for a set of users U. These resources may, for example, be data objects, permissions (typically modeled as an object-action pair, as in RBAC96 [14]), or steps in a workflow [2]. We follow existing work on resiliency policies by assuming an authorization policy is specified as a user-resource relation $UR \subseteq U \times R$, where $(u, r) \in UR$ means that user u is authorized to access resource r.[1]

For a user $u \in U$, we write $N(u)$ to denote $\{r \in R : (u, r) \in UR\}$; that is, $N(u)$ is the set of resources for which u is authorized, and is called the *neighborhood*[2] of u. We extend this notation so that $N(V)$, for any $V \subseteq U$, denotes $\bigcup_{u \in V} N(u)$; that is, $N(V)$, called the *neighborhood* of V, is the set of resources for which users in V, collectively, are authorized. For any integer $q \in \mathbb{N}$, we will write $[q]$ to denote $\{1, \dots, q\}$.

[1]Clearly, more complex policies may be specified, but invariably such policies may be reduced to such a relation in polynomial time.

[2]We use the term neighborhood as we view UR as a bipartite graph; this view may help the reader, too.

2.1 Resiliency policies

Informally, resiliency in the context of access control is related to fault tolerance. It refers to the ability of a multi-user system, governed by some access policy, to continue functioning if authorized users are unavailable. In order to formalize the notion of resiliency and to study its properties, Li, Wang and Tripunitara [13] introduce the concept of a *resiliency policy*, which has the form $\mathsf{res}(P, s, d, t)$, where P is a subset of R, and s, d and t are integers such that $s \geqslant 0$, $d \geqslant 1$ and $t \geqslant 1$. The satisfaction of policy $\mathsf{res}(P, s, d, t)$ is determined in the context of a user-resource authorization relation UR.

Definition 1. *An authorization relation UR* satisfies $\mathsf{res}(P, s, d, t)$ *if and only if upon removal of any set of s users, there exist d mutually disjoint sets of users $V_1, \dots, V_d$ such that $N(V_i) \supseteq P$ and $|V_i| \leqslant t$. We call such a set $\{V_1, \dots, V_d\}$ a* solution/set of teams.

Informally, the resiliency policy $\mathsf{res}(P, s, d, t)$ specifies a requirement that the access control state UR should be able to tolerate the absence of at most s users and still have sufficient users authorized for some critical task (associated with the resources in P). In the most general version, the critical task requires d teams of users, each of size at most t, such that each team independently has all authorizations for resources in P. Li *et al.* then define the following decision problem [13, Definition 2].

Definition 2. *Given a user-resource relation UR and a resiliency policy $\mathsf{res}(P, s, d, t)$, the* Resiliency Checking Problem *(*RCP*) asks whether UR satisfies $\mathsf{res}(P, s, d, t)$.*

Li *et al.* introduce bracket notation RCP$\langle\rangle$ to denote some restrictions of the problem, in which one or more parameters (among s, d and t) are fixed: s and d can respectively be set to 0 and/or 1 (or other fixed positive values), while t can be set to infinity (∞), meaning that there is no constraint on the size of the sets. Note that setting $t = \infty$ is equivalent to setting $t = |P|$, as the size of any suitable team contains at most $|P|$ authorized users. Henceforth, we write p to denote $|P|$.

As Li *et al.* [13] point out, RCP$\langle\rangle$ and its special cases are connected to several well-known combinatorial problems such as the Set Cover and Domatic Partition problems. They were thus able to establish several, mainly negative, results concerning the computational complexity of RCP$\langle\rangle$, summarized by the following result [13, Theorem 1].

Theorem 1. *We have the following:*

- RCP$\langle\rangle$, RCP$\langle d = 1\rangle$ *and* RCP$\langle t = \infty\rangle$ *are NP-hard and are in coNP^{NP}.*
- RCP$\langle s = 0, d = 1\rangle$, RCP$\langle s = 0, t = \infty\rangle$ *and* RCP$\langle s = 0\rangle$ *are NP-complete.*
- RCP$\langle d = 1, t = \infty\rangle$ *can be solved in linear time.*

In practice, it seems reasonable to assume that s will be much smaller than the total number of users. This amounts to saying that at any given time only a very small percentage of users will be unvailable. Similarly, in many settings, it seems reasonable to assume that d will also be much smaller than the total number of users. Indeed, in many settings, the case $d = 1$ will be the only one of interest.

Given the relative sizes of the parameters, in particular the sharp disparity between the number of users and the other parameters, it is worth investigating the extent to which the complexity of RCP$\langle\rangle$ is influenced by each of the parameters. For this reason, we will analyze RCP$\langle\rangle$ using the techniques from parameterized complexity, also called multivariate complexity analysis [7].

2.2 Workflow satisfiability

A workflow is a collection of steps that must be executed by some users in order to achieve an objective. The order in which those steps should be executed is usually given, as well as constraints and authorization policies, restricting the users that can be assigned to steps [2, 15].

More formally, a *workflow specification* is defined by a directed, acyclic graph $G = (S, E)$, where S is a set of steps and $E \subseteq S \times S$. Given a workflow specification (S, E) and a set of users U, an *authorization policy* for a workflow specification is a relation $A \subseteq S \times U$; we say user u is *authorized* to perform step t if $(t, u) \in A$. A *workflow constraint* has the form (T, Θ), where $T \subseteq S$ and Θ is a family of functions with domain T and range U; we say T is the *scope* of the constraint (T, Θ). (Informally, Θ defines the set of authorized mappings from steps in T to users in U. Generally speaking, the elements of Θ are not explicitly enumerated; we discuss this in more detail below.) Then a *constrained workflow authorization schema* is a pair $((S, E), U, A, C)$, where (S, E) is a workflow specification, U is a set of users, A is an authorization policy, and C is a set of workflow constraints.

Once a constrained workflow authorization schema has been defined, the goal is to assign a user to each step, so that all constraints and the authorization policy are satisfied. More formally, a (partial) *plan* is a function $\pi : T \to U$, where $T \subseteq S$. A plan π is *complete* if $T = S$. Given a plan $\pi : T \to U$ and $T' \subseteq T$, we write $\pi|_{T'} : T' \to U$ to denote the plan such that $\pi|_{T'}(t) = \pi(t)$ for all $t \in T'$. We say a plan $\pi : S' \to U$ *satisfies* a workflow constraint (T, Θ) if $T \not\subseteq S'$ or $\pi|_T \in \Theta$. (Informally, if the range of π includes the scope of a constraint, then the restriction of π to T must be one of the authorized mappings.) Finally, we say a plan $\pi : S \to U$ is *valid* if it satisfies every constraint in C and, for all $t \in S$, $(t, \pi(t)) \in A$. The combination of authorization policy and constraints may mean that no valid plan exists, motivating the following problem.

Definition 3. *Given a constrained workflow authorization schema $W = ((S, E), U, A, C)$, the* Workflow Satisfiability Problem *(*WSP*) asks whether there exists a valid plan for W.*

In practice, we do not define constraints by enumerating all possible elements of Θ. Instead, we define different families of constraints that have "compact" descriptions. For instance, two simple constraints we will use in this paper are the atleast and atmost constraints. Both have two arguments: the scope $P \subseteq S$ to which they apply, and an integer $\ell \leq |U|$. Then, a plan $\pi : S \to U$ satisfies $\mathsf{atleast}(P, \ell)$ (resp. $\mathsf{atmost}(P, \ell)$) if and only if $|\pi(P)| \geq \ell$ (resp. $|\pi(P)| \leq \ell$), that is, if the steps in P are assigned to at least (resp. at most) ℓ users.

There now exists a substantial body of work on constraints in workflow systems, beginning with the seminal work of Bertino, Ferrari and Atluri [2]. In particular, separation/binding of duty constraints and cardinality constraints are recognized as being important and have been widely studied. Informally, atleast constraints generalize separation of duty constraints, and are analogous to the ssod constraints studied by Li *et al.* [13], while atmost constraints generalize binding-of-duty constraints. In particular, $\mathsf{atleast}(\{t, t'\}, 2)$ requires that steps t and t' are performed by separate users, while $\mathsf{atmost}(\{t, t'\}, 1)$ requires that steps t and t' are performed by the same user.

User-independent constraints generalize all these forms of constraints [3]. Informally, such a constraint limits the execution of steps in a workflow, but is indifferent to the particular users that execute the steps. More formally, a constraint (T, Θ) is user-independent if whenever $\theta \in \Theta$ and $\psi : U \to U$ is a permutation then $\psi \circ \theta \in \Theta$ (where $\circ$ denotes function composition). A separation of duty constraint, on two steps for example, simply requires that two *different* users execute the steps, not that, say, Alice and Bob must execute them. Similarly, a binding of duty constraint on two steps only requires that the *same* user executes the steps. More generally, atleast and atmost constraints are user-independent. It appears most constraints that are useful in practice are user-independent: all constraints defined in the ANSI-RBAC standard [1], for example, are user-independent.

2.3 Fixed-parameter tractability

It is generally assumed that no polynomial-time algorithm exists to solve an NP-hard decision problem. In other words, the running-time of any algorithm for solving such a problem is exponential in the size of the input to the problem. However, many decision problems have multiple parameters and the relative magnitude of the values taken by those parameters might be quite different in most, if not all, conceivable practical instances of the problem. Hence, it is worth considering whether there are algorithms whose running-time is exponential in the parameters expected to take small values (and polynomial in the size of the input). It is this approach we will adopt in the study of this paper.

Formally, we say that a decision problem is *fixed-parameter tractable* (FPT) if there exists an algorithm that decides if an instance is positive in $O(f(k)p(n)) = O^*(f(k))$ time[3] for some computable function f and some polynomial p, where n denotes the size of an instance, and is k a parameter (or a combination of several parameters) of the instance. Accordingly, we will call such an algorithm an *FPT algorithm.* Multivariate analysis of the complexity of a problem may identify an FPT algorithm whose running time is exponential in specific parameters that are known to be small in practice, leading to the development of efficient algorithms for problem instances of practical interest. For more details about parameterized complexity, we refer the reader to the monograph of Downey and Fellows [7].

In the particular case of WSP, a naive algorithm to solve the problem considers every possible plan in turn. Assum-

[3]We will follow the convention adopted in the literature on FPT algorithms [7] by using the $O^*(.)$ notation, which omits polynomial terms and factors.

ing we can test whether a plan is valid in time polynomial in $n = |U|$ and $k = |S|$, the algorithm has running time $O^*(n^k)$. Wang and Li [15] were the first to observe that the number of users n is typically an order of magnitude larger than the number of steps k. They used this observation to show that WSP is FPT when k is the parameter and all constraints are separation-of-duty or binding-of-duty constraints [15]. Recent work has significantly extended Wang and Li's results, in terms of the running times for FPT algorithms and the range of constraints that can be included in the workflow specification. The most powerful theoretical results relate to user-independent constraints [3] and extensive experimental work has shown that implementations of the resulting FPT algorithms have practical value [3, 11].

3. THE RESILIENCY CHECKING PROBLEM

We first argue that the main interest in designing efficient algorithms for RCP$\langle\rangle$, either from the practical or theoretical point of view, lies in the case where $s = 0$. Notice that all the following ideas also apply to the variant RCP$\langle t = \infty\rangle$ (mainly because, as we saw previously, we may assume that $t = p$).

We first review the method used by Li *et al.* [13] to solve RCP$\langle\rangle$ (and RCP$\langle t = \infty\rangle$). Given a resiliency policy $\mathsf{res}(P, s, d, t)$ and a relation UR, their initial idea is to enumerate all subsets of at most s users, and, for each such subset V, to test whether the resiliency policy $\mathsf{res}(P, 0, d, t)$ is satisfied by the instance obtained by removing the users in V from U (and deleting the corresponding elements from UR accordingly). Then, using a domination argument, they observe that the exhaustive enumeration of all such subsets can be avoided, and define two kinds of pruning strategies. The satisfaction of the resulting instance (in which $s = 0$) is then determined by translating the problem into a SAT instance and using SAT4J, an off-the-shelf SAT solver, to solve it. In their experiments, they observe that their *static pruning* strategy is quite efficient, and that the bottleneck of their algorithm seems to be the satisfiability test for the $\mathsf{res}(P, 0, d, t)$ policy on the reduced instance. From this observation, it makes sense to focus on the resolution of RCP$\langle s = 0\rangle$ only.

Another argument for this approach is the parameterized complexity of RCP$\langle\rangle$. As we mentioned earlier, it makes sense to analyse the performance of our algorithms for RCP$\langle\rangle$ and its variants using parameters such as $p = |P|$, s, d and t. The following result asserts that RCP$\langle\rangle$ can be tackled by solving $2^{s\log(dt)}$ instances of RCP$\langle s = 0\rangle$.[4] This result will let us focus on the development of an efficient algorithm for RCP$\langle s = 0\rangle$, and more precisely of a fixed-parameter tractable algorithm parameterized by (p, d) (Theorem 3). Combining these two results, we will be able to show that RCP$\langle\rangle$ is itself fixed-parameter tractable parameterized by (p, s, d, t).

THEOREM 2. *Suppose that there is an algorithm for* RCP$\langle s = 0\rangle$ *which returns a set of teams in case of a satisfiable instance (and answers no in case of an unsatisfiable one) in time $O^*(f(p, d, t))$ for some computable function f. Then* RCP$\langle\rangle$ *can be solved in $O^*(2^{s\log(dt)} f(p, d, t))$ time.*

[4]All logarithms used in this paper are in base 2, unless otherwise stated.

In summary, using RCP$\langle s = 0\rangle$ as a black box for solving the more general case makes sense both from a practical and theoretical point of view. In the remainder of this section, we thus focus on RCP$\langle s = 0\rangle$ only, and show how the workflow satisfiability problem can be used to solve the problem as well as several generalizations of it which might be of practical interest.

3.1 RCP as a workflow satisfiability problem

We first discuss how an instance of RCP$\langle s = 0\rangle$ may be transformed into an instance of WSP with user-independent constraints. We then show that it is possible to define several interesting generalizations of RCP$\langle\rangle$ and to use a similar transformation to WSP to solve these problems.

3.1.1 Reduction

The intuition behind our reduction from RCP$\langle\rangle$ to WSP lies in the observation that RCP$\langle\rangle$ asks for a mapping from d copies of a resource to d different authorized users, while WSP asks whether there exists a mapping from steps to authorized users, subject to some constraints. Hence, we translate the set of resources in RCP$\langle\rangle$ into an appropriate set of steps and the constraints imposed on the structure of the teams by a resiliency policy to user-independent workflow constraints.

CONSTRUCTION 1. *Given $UR \subseteq U \times R$ and $\mathsf{res}(P, 0, d, t)$ with $P \subseteq R$, we construct a constrained workflow authorization scheme W in which $E = \emptyset$ and S comprises d copies of P. We write these distinct sets as $P^1, \ldots, P^d$ and, for $r \in P$, we write r^i to denote the copy of r in set P^i. Then we define:*

$$A = \bigcup_{i=1}^{d} \{(u, r^i) : (u, r) \in UR\};$$
$$C = \{\mathsf{atleast}(\{r^i, s^j\}, 2) : r, s \in P, 1 \le i < j \le d\} \cup \{\mathsf{atmost}(P^i, t) : i \in [d]\}.$$

LEMMA 1. *$\mathsf{res}(P, 0, d, t)$ is satisfied by UR if and only if W is satisfiable.*

PROOF. Suppose that UR satisfies $\mathsf{res}(P, 0, d, t)$, *i.e.* there exists a set of teams $\{V_1, \ldots, V_d\}$. Since $N(V_i) \supseteq P$ for all $i \in [d]$, and since all teams are pairwise disjoint, for every $r \in P$ there exists $(u_1^r, \ldots, u_d^r) \in V_1 \times \cdots \times V_d$ such that $(u_i, r) \in UR$ for all $i \in [d]$. By definition of A, observe that we can construct a plan $\pi : P^1 \cup \cdots \cup P^d \to U$, by setting, for all $i \in [d]$ and all $r \in P^i$, $\pi(r) = u_i^{r_i}$. This plan will not violate any $\mathsf{atleast}$ constraint by definition, and will not violate any atmost constraint since $|V_i| \le t$ for all $i \in [d]$.

Conversely, suppose that W is satisfiable, *i.e.* there exists a valid plan $\pi : P^1 \cup \cdots \cup P^d \to U$. For all $i \in [d]$, we define $\pi_i : P^i \to U$ by $\pi_i(r) = \pi(r)$ for all $r \in P^i$, and let $V_i = \pi_i(P^i)$. Since π is a valid plan, the atmost constraints ensure that $|V_i| \le t$ while the $\mathsf{atleast}$ constraints ensure that all V_i are pairwise disjoint. Finally, by construction we have $N(V_i) \supseteq P$, which proves that $\mathsf{res}(P, 0, d, t)$ is satisfied. $\square$

Note that if $d = 1$ (meaning we require a single team of authorized users), then we have a simple instance of WSP, where the set of steps corresponds exactly to P. Note also that if $t = \infty$ (or indeed $t = p$), then we do not require any atmost constraints.

3.1.2 Algorithm

There has been a considerable amount of research into FPT algorithms for WSP in recent years. In particular, recent work by Cohen *et al.* [3] and Karapetyan *et al.* [11] has led to optimized and demonstrably efficient algorithms for WSP (in the case that k is considerably smaller than n).

The instances of WSP obtained from instances of RCP$\langle\rangle$ have a particular form, because of the specific constraints a resiliency policy imposes. In this section, we improve the running-time $O^*(2^{k\log(k)}) = O^*(2^{dp\log(dp)})$ of the Pattern Backtracking algorithm [11], exploiting the particular nature of the WSP instances obtained from RCP$\langle\rangle$.

THEOREM 3. RCP$\langle s=0\rangle$ *can be solved in* $O^*(2^{dp\log(p)})$.

PROOF. As described in Construction 1, we are given a set S of steps composed of d pairwise disjoint sets $P^1,\ldots,P^d$, each containing p steps, an authorization policy A, and a set of atleast and atmost constraints. In particular, the scope of each atleast constraint lies in two different sets P^i and P^j, while the scope of each atmost constraint lies in a unique set P^i.

Given a plan $\pi: S' \to U$, where $S' \subseteq S$, the *pattern* $Pat(\pi)$ of π is the partition $\{\pi^{-1}(u):\ u\in U,\ \pi^{-1}(u)\neq\emptyset\}$ of S' into non-empty sets. We say that two plans π and π' are *equivalent* if they have the same pattern. A pattern is said to be *complete* if $S'=S$. We say that a pattern $T=\{V_1,\ldots,V_{|T|}\}$ is *valid* if there exists a valid plan $\pi: S\to U$ such that $Pat(\pi)=T$. Observe that given a pattern $T=\{V_1,\ldots,V_{|T|}\}$ of S, we can, in polynomial time, decide whether it is valid pattern, and to find in that case a valid plan, by finding a perfect matching in an appropriate bipartite graph (namely, with partite sets $\{1,\ldots,|T|\}$ and U, and an edge between $i\in\{1,\ldots,|T|\}$ and $u\in U$ whenever there exists $r\in V_i$ such that $(u,r)\in A$). Hence, the algorithm is a procedure for finding a valid pattern.

A naive upper bound on the number of all possible patterns of S is the number of all partitions of S into non-empty sets; that is, the $|S|$-th Bell number, $B_{|S|}=O(2^{dp\log(dp)})$. However, it can be seen that in our case, any pattern T containing V such that $V\cap P^i\neq\emptyset$ and $V\cap P^j\neq\emptyset$ for $i\neq j$, cannot be valid, as otherwise it would violate at least one atleast constraint. Thus, the number of valid patterns can actually be reduced to $(B_p)^d = O(2^{dp\log(p)})$, which proves the claimed running time. $\square$

Combining Theorems 2 and 3, we obtain the following result:

THEOREM 4. RCP$\langle\rangle$ *(and* RCP$\langle t=\infty\rangle$*) is fixed-parameter tractable parameterized by* (p,s,d,t).

3.1.3 Discussion

The Pattern Backtracking algorithm for WSP has obtained good results in practice [11]. The experiments conducted by Karapetyan *et al.* used values of k between 10 and 65, together with varying numbers of (atleast and atmost) constraints, and defined $n=10k$. The performance of the Pattern Backtracking algorithm was compared to that of SAT4J, a state-of-the-art off-the-shelf SAT solver, running on WSP instances transformed into pseudo-Boolean SAT instances [11, 15]. The Pattern Backtracking algorithm was able to solve all WSP instances containing less than 60 steps, in contrast to the SAT solver, which was rarely able to find solutions for instances containing more than 25 steps.

The small parameter in WSP is the number k of workflow steps and in the reduction from RCP$\langle\rangle$ to WSP, the number of workflow steps is dp. The running-time of our algorithm is exponential in $dp\log p$, so we require this parameter to be relatively small for this reduction to be of practical use. In the case $d=1$, we expect to solve instances of RCP in which P is relatively large (up to 60). If $d>1$, we will obviously need a corresponding reduction in the size of p.

Hence, we believe that our reduction to WSP and use of the Pattern Backtracking algorithm is of practical value, and will signicantly extend the size of the RCP instances that can currently be solved [13]. With the above observations in mind, we recall the experimental results presented by Li *et al.* [13, Figure 3]. In their experiments, the parameters $s=3$ and $p=10$ were fixed, while d varied between 2 and 7, and n varied between 40 and 100. Their approach involved two steps: the first reduced an instance of RCP$\langle s=3\rangle$ to multiple instances of RCP$\langle s=0\rangle$; the second solved the instance of RCP$\langle s=0\rangle$ by translating it to an instance of SAT and solving the SAT instance using SAT4J. The static pruning technique used to perform the first step meant that SAT4J had to solve approximately the same number of instances of RCP for a given value of s (independent of the number of users n) [13, Table I]. This means, that the difference in running times on instances of RCP$\langle s=3\rangle$ for a given n is largely determined by d (since p is fixed). Li *et al.* noted that the performance of the SAT solver (on instances of RCP$\langle s=0\rangle$) began to degrade significantly as d increased, and do not report results for d larger than 7. We believe that using the Pattern Backtracking algorithm in step two would mean that far larger instances of RCP could be solved. In particular, we could significantly increase the number of users (up to several hundred). Moreover, we could solve a wider range of instances: we could, for example, solve instances in which $d>7$, provided there was a corresponding reduction in p.

Our multivariate analysis of RCP suggests that instances of RCP$\langle s=0\rangle$ can be solved in a reasonable amount of time, provided $dp\log p$ is relatively small. Moreover, as we will see in the next section, our approach to solving RCP is rather flexible, in the sense that we can impose additional requirements on the solution to RCP and, with very simple modifications to Construction 1, translate the problem to an instance of WSP. This is unlikely to be the case for the methods used by Li *et al.* to solve RCP, which involved creating a specific SAT encoding for RCP and the use of a SAT solver; different versions of RCP will require new encodings.

3.2 Generalizing resiliency policies

We now describe how the model of resiliency of Li *et al.* [13] can be generalized in order to capture some other situations that might occur in practice. For each of these cases, we demonstrate that a simple adaptation of Construction 1 enables us to model the problem as an instance of WSP: generally speaking, it is simply a matter of adjusting the set of constraints. As we saw in the beginning of Section 3, it makes sense to focus on the problem of testing the satisfaction of the given property only (the case $s=0$), rather than its "resiliency" part. That is why only this subcase will be considered in the following sub-sections.

3.2.1 Adding counting constraints

The original definition of a resiliency policy imposes constraints on the composition of the teams, by insisting on disjointness and that each team has at most t users. It is debatable whether disjointness is a property that would be important in practice, and we relax this assumption in Section 3.2.2. However, we first consider, two, perhaps more obvious, constraints on the composition of the teams, and show that we can modify Construction 1 to produce an instance of WSP.

First, we might consider imposing another "local" constraint on team composition, requiring that at least ℓ users belong to each team. Such a requirement might be imposed in the interests of separation of duty or to ensure a reasonably fair division of labor. In particular, we might set $\ell = t$, in which case every team would have exactly t users. Of course, in the context of Construction 1, this simply means including the following (additional) constraints:

$$\left\{ \mathsf{atleast}(P^i, \ell) : i \in [d] \right\}.$$

A further possibility is to impose different constraints on the size of each team. (This simply requires a different value ℓ_i to be associated with each P^i in the $\mathsf{atleast}$ constraints.)

Second, we might consider imposing "global" constraints. The effect of insisting that each team has at most t users is that the total number of users is no greater than dt. However, we may wish to impose a global upper bound $L_{\max}$ (less than dt) on the total number of users in the d teams. We can achieve this simply including the constraint $\mathsf{atmost}(S, L_{\max})$ in the WSP formulation. Similarly, we can impose a global lower bound $L_{\min}$ on the total number of users, which we can achieve by including the constraint $\mathsf{atleast}(S, L_{\min})$ in the WSP formulation.

The most important observation to make here is that all of these constraints on team composition are *user-independent* constraints in the WSP instance.

3.2.2 Allowing team overlaps

In their definition of $\mathsf{res}(P, s, d, t)$, Li *et al.* [13] insist on the disjointness of the d teams. We may relax this condition and consider the possibility of allowing teams to overlap. (In the context of workflow systems, this would mean that a user could participate in the execution of more than one workflow instance.) Hence, we introduce an extended form of resiliency policy $\mathsf{xres}(P, s, d, d', t)$, with $d' \leqslant d$. Such a policy is satisfied if, for all subsets V of size s in U, there exist d sets $V_1, \ldots, V_d$ of users such that for all $i \in [d]$ and all $u \in U$: (i) $V_i \cap V = \emptyset$; (ii) u belongs to at most d' of $V_1, \ldots, V_d$; (iii) $|V_i| \leqslant t$; (iv) $N(V_i) \supseteq P$. The condition that u belongs to at most d' teams seeks to limit the workload of each user. Note that setting $d' = d$ imposes no upper limit on the number of teams to which any one user can belong, while $d' = 1$ corresponds to the "standard" resiliency policy $\mathsf{res}(P, s, d, t)$, and thus the satisfaction of this new policy generalizes the one of Definition 1.

We now focus on the problem of testing the satisfaction of a dynamic resiliency policy $\mathsf{xres}(P, 0, d, d', t)$ (*i.e.* with $s = 0$). As noted previously, similar arguments as for RCP$\langle\rangle$ can be made for this new problem, and the case $s = 0$ can be used as a black-box to solve the more general one.

We introduce a construction to transform the satisfaction of $\mathsf{xres}(P, 0, d, d', t)$ into an instance of WSP. We no longer require that a user belonging to a "team" executing steps in one workflow instance cannot belong to a team executing a different instance. Hence, we no longer require the $\mathsf{atleast}$ constraint that we used in Construction 1. Instead, we have to impose constraints on the total number of steps to which any one user is assigned. As in Construction 1, the set of steps will be defined by $S = P^1 \cup \ldots P^d$, where P^i is a copy of P, for all $i \in [d]$, and the authorization policy is defined to be $A = \bigcup_{i=1}^{d} \{(u, r^i) : (u, r) \in UR\}$ (recall that r^i denotes the copy of r in P^i, for all $r \in P$ and $i \in [d]$). Then, we have to ensure that for any set of $(d' + 1)$ steps, each chosen in a different P^i, the number of users assigned to these steps is at least 2. Accordingly, we define

$$\zeta(S) = \{X \subseteq S : |X| = d' + 1, |P^i \cap X| \leq 1, i \in [d]\}.$$

Finally, determining the satisfaction of an $\mathsf{xres}(P, 0, d, d', t)$ policy can be done by replacing the set of constraints of Construction 1 by

$$C = \{\mathsf{atleast}(X, 2) : X \in \zeta(S)\} \cup \{\mathsf{atmost}(P^i, t) : i \in [d]\}$$

and solving the resulting instance of WSP, using *e.g.* the Pattern Backtracking algorithm of [11] (note that the optimization of the algorithm provided in Theorem 3 can no longer be applied to the WSP instance obtained in this particular case).

Note, however, that the WSP instance we now solve contains a number of constraints exponential in d and p, implying an additional computational cost when solving this new problem, as one could expect. Observe that there are exactly $|\zeta(S)| + d$ constraints, and that

$$|\zeta(S)| = O\left(\binom{d}{d'+1} p^{d'+1} \right) = O\left(2^{(d'+1)(\log(d)+\log(p))} \right),$$

since for each $I \subseteq \{1, \ldots, d\}$ of size $d' + 1$, there are at most $p^{d'+1}$ subsets of S intersecting P^i in exactly one element, for all $i \in I$.

It is worth noting that the set of constraints obtained from translating an instance of RCP$\langle\rangle$ in which $d' = d$ (that is, when we impose no upper limit on the number of teams to which a user can belong) is simply

$$C = \{\mathsf{atmost}(P^i, t) : i \in [d]\}.$$

In other words, if we are not concerned with individual workloads, the WSP instance for the version of RCP in which teams need not be disjoint is actually easier than the original version of RCP$\langle\rangle$.

Table 1 summarizes the possible requirements for team composition in the context of resiliency and associates those requirements with the corresponding constraints in a WSP instance. Selecting requirements 1 and 2, for example, corresponds to the standard resiliency policies of Li *et al.* Selecting requirements 1, 2 and 3 would insist that all teams have exactly t members and every team is disjoint. In the next two subsections, we consider further possibilities for generalizing RCP.

3.2.3 Independent teams

In the definition of RCP$\langle\rangle$, all d teams are required to be of size at most t, and are allocated to the same set of resources $P \subseteq R$. A natural generalization is to allow each team to have a different size, and to be associated with a different set of resources.

Resiliency requirement	Constraint(s) in WSP
1. Teams should have no more than t users	$\{\mathsf{atmost}(P^i, t) : i \in [d]\}$
2. Teams should be disjoint	$\{\mathsf{atleast}(\{r^i, s^j\}, 2) : r, s \in P, 1 \leq i < j \leq d\}$
3. Teams should have at least ℓ users	$\{\mathsf{atleast}(P^i, \ell) : i \in [d]\}$
4. Teams should contain no more than $L_{\max}$ users in total	$\{\mathsf{atmost}(P^1 \cup \cdots \cup P^d, L_{\max})\}$
5. Teams should contain at least $L_{\min}$ users in total	$\{\mathsf{atleast}(P^1 \cup \cdots \cup P^d, L_{\min})\}$
6. Each user should belong to no more than d' teams	$\{\mathsf{atleast}(X, d') : X \subseteq S, \lvert X\rvert = d'+1, \lvert P^i \cap X\rvert \leq 1, i \in [d]\}$

Table 1: Summary of possible resiliency policies

First, we define a *team-independent* constraint $\mathsf{t\text{-}ind}(P, t)$, which is satisfied by $UR \subseteq U \times R$ if and only if there exists a set containing no more than t users who are collectively authorized for P. Then, a *team-independent resiliency* constraint is given by $\mathsf{ti\text{-}res}(s, P_1, t_1, \ldots, P_d, t_d)$, and is satisfied by $UR \subseteq U \times R$ if and only if on removal of any set of s users, the team-independent constraints $\mathsf{t\text{-}ind}(P_1, t_1), \ldots, \mathsf{t\text{-}ind}(P_d, t_d)$ are all satisfied by disjoint sets of users.

In particular, $\mathsf{res}(P, s, d, t)$ (in the sense of Definition 1) is a team-independent resiliency constraint in which the scope and team size of each $\mathsf{t\text{-}ind}(P_i, t_i)$ constraint are P and t, respectively. Hence, this new problem is also a generalization of RCP$\langle\rangle$.

Here again, we modify Construction 1 in order to test the satisfaction of $\mathsf{ti\text{-}res}(0, P_1, t_1, \ldots, P_d, t_d)$. We define the set of steps to be $S = P^1 \cup \cdots \cup P^d$, where P^i is a copy of P_i for all $i \in [d]$. Again, for all $r \in R$, we denote by r^i its copy in P^i, in the case where $r \in P_i$. Then the authorization policy remains $A = \bigcup_{i=1}^{d} \{(u, r^i) : (u, r) \in UR \text{ and } r \in P_i\}$. Finally, we define the set of constraints $C = C_1 \cup C_2$, with:

$$C_1 = \{\mathsf{atmost}(P^i, t_i) : i \in [d]\};$$
$$C_2 = \{\mathsf{atleast}(\{r^i, s^j\}, 2) : r \in P^i, s \in P^j, 1 \leq i < j \leq d\}.$$

Here again, the Pattern Backtracking algorithm of Theorem 3 can be used to solve the obtained instance of WSP. Thus, the problem of testing whether an authorization policy satisfies $\mathsf{ti\text{-}res}(s, P_1, t_1, \ldots, P_d, t_d)$ is FPT parameterized by $\lvert \bigcup_{i=1}^{d} P_i \rvert$, s, d, and $\max_{i=1\ldots d} t_i$.

Naturally, we can also define an extended version of a team-independent resiliency constraint, in which we relax the condition that the teams be disjoint (as in Section 3.2.2). Again, we may distinguish between the case when there is an upper limit on the number of teams to which a user may belong ($d' < d$), where we will require a large number of additional constraints, and the case where no such limit is imposed ($d' = d$), and the number of constraints is actually reduced.

3.2.4 *Weighted resiliency*

In Section 3.2.1, we considered the possibility of imposing a global upper limit on the total number of users in the teams. In practice, it might be the case that different users "cost" a different amount. We might imagine, for example, departments charging out the use of its employees by other departments, and that more senior staff cost more than junior staff. In other words, as well as imposing an upper limit on the total number of users, we might consider associating each user with a cost and imposing an upper limit – a budget B – on the total cost of the users. Accordingly, we introduce a cost function defined over the set of users $c : U \to \mathbb{Z}$. This weighted variant of RCP$\langle\rangle$ then asks for teams $V_1, \ldots, V_d$ such that $\lvert V_i\rvert \leq t$ for all $i \in [d]$ and

$$\sum_{i=1}^{d} \sum_{u \in V_i} c(u)$$

is minimum over all possible sets of teams.

In order to deal with this variant, we use a recent generalization of WSP to a weighted version [4], called VALUED-WORKFLOW SATISFACTION PROBLEM (V-WSP). This problem allows us to define, for each set of steps $T \subseteq S$ and user $u \in U$, a weight $w(T, u)$ representing the cost of assigning u to steps in T. The goal is then to find a valid plan $\pi : S \to U$ such that

$$w_A(\pi) = \sum_{u \in U} w(\pi^{-1}(u), u)$$

is minimum over all valid plans[5]. Hence, we use once again Construction 1, and set, for every $u \in U$ and all $T \subseteq P^i$ for all $i \in [d]$:

$$w(T, u) = \begin{cases} c(u) & \text{if } T \subseteq N(u), \\ \infty & \text{otherwise.} \end{cases}$$

Using similar ideas as in the proof of Lemma 1, the algorithm for V-WSP will output d plans $\pi_i : P^i \to U$, $i \in [d]$, defining the teams $V_1, \ldots, V_d$. As explained earlier, this plan will be such that

$$\sum_{i=1}^{d} \sum_{u \in V_i} w(\pi^{-1}(u), u) = \sum_{i=1}^{d} \sum_{u \in V_i} c(u)$$

is minimum, over all plans not violating any constraint. It is worth pointing out that the algorithm for V-WSP of [4] can also be improved in the same way as described in the proof of Theorem 3.

Furthermore, still using the particular structure of the obtained V-WSP instance, it is also possible to change the algorithm so that it outputs a set of teams $V_1, \ldots, V_d$ such that $\sum_{u \in V_i} c(u) \leq W_i$ for some given bounds $W_1, \ldots, W_d$, if such a set exists. In other words, we now ask in this variant that each team *independently* respect its own upper bound. A possible application of this variant might be as follows: suppose that the set of users contains a subset U_s of special users, and the task that needs to be achieved (by having

[5]Notice that in the definition of V-WSP of [4], a weight function over the constraints w_C is also defined, and the goal is actually to find a plan which minimizes $w_A(\pi) + w_C(\pi)$. However, assigning an infinite cost to each constraint simply forbids the violation of any constraint and forces an optimal solution to minimize $w_A(\pi)$ over all plans with a finite constraint weight, as desired here.

access to all resources in P) necessarily requires the presence of at least one special user. For instance, one might think of the special users as managers, or team leaders. This problem can simply be solved using the aforementioned method by setting $c(u) = -t + 1$ for all $u \in U_s$, $c(u) = 1$ for all $u \in U \setminus U_s$, and setting $W_i = 0$ for all $i \in [d]$.

4. STATIC SEPARATION-OF-DUTY AND RESILIENCY

A *static separation of duty* (SSoD) *policy* has the form $\mathsf{ssod}(P, t)$, where P is a subset of the set of resources R and t is an integer such that $1 < t \leqslant |P|$ [13]. Satisfaction of this constraint is determined by considering the authorization policy UR (and we ensure continuing satisfaction of the constraint by controlling updates to UR). More formally, the SSoD policy $\mathsf{ssod}(P, t)$ is *satisfied* by $UR \subseteq U \times R$ if and only if for all subsets V of U such that $|V| < t$, $P \not\subseteq N(V)$. So, for example, the static separation-of-duty constraint $\mathsf{ssod}(\{p_1, p_2\}, 2)$, requires that two particular resources are not assigned to the same user.

Equivalently, $\mathsf{ssod}(P, t)$ is *violated* if there exists a set of at most $t - 1$ users V such that $N(V) \supseteq P$. A naive algorithm for testing the satisfiability of an $\mathsf{ssod}(P, t)$ policy would be to check if the neighborhood of some subset of at most $t - 1$ users contains P. However, such an algorithm would necessarily be of complexity $O^*(n^{t-1})$, which would be inefficient in practice. Thus, our goal is to obtain an FPT algorithm parameterized by some smaller parameters, like $p = |P|$. To do so, we now explain how the problem of determining whether a static separation-of-duty policy is satisfied may be converted into an instance of WSP.

Construction 2. *Given $UR \subseteq U \times R$ and $\mathsf{ssod}(P, \ell)$, we construct a workflow specification W in which $S = P$, $E = \emptyset$, $A = UR \cap (U \times P)$, and $C = \{\mathsf{atmost}(P, t - 1)\}$.*

Lemma 2. *$\mathsf{ssod}(P, t)$ is satisfied if and only if W is unsatisfiable.*

Note that $\mathsf{atmost}(P, t)$ is a user-independent constraint, and that the number of steps in the obtained WSP instance is $p = |P|$. Thus, here again the Pattern Backtracking algorithm for WSP [11] gives a practical algorithm to test whether a static separation-of-duty constraint is satisfied by an authorization policy, in a worst-case $O^*(2^{p\log(p)})$ time and polynomial space. In addition, as Li *et al.* [13] notice, the complement of this problem is actually equivalent to the Set Cover problem[6]. Known positive [8] and negative [6] results for Set Cover lead to the following.

Theorem 5. *Testing whether a static separation-of-duty $\mathsf{ssod}(P, t)$ is satisfied by an authorization policy UR can be done in $O^*(2^p)$ time, while a $O^*(2^{o(p)})$ algorithm would violate Exponential Time Hypothesis*[7].

Notice that the $O^*(2^p)$ algorithm of [8] relies on a dynamic programming approach, thus using exponential space.

[6]In the Set Cover problem, we are given a set $\mathcal{S}$ of subsets of a set U, and an integer k, and the aim is to find at most k elements of $\mathcal{S}$ whose union is U.

[7]The Exponential Time Hypothesis is the assumption that 3-SAT cannot be solved in time $O^*(2^{o(n)})$, where n is the number of variables in the CNF formula [10].

Hence, we strongly believe that applying the Pattern Backtracking algorithm on the instance obtained by Construction 2 is likely to be more efficient in practice, although having a worse theoretical running time ($O^*(2^{p\log(p)})$).

4.1 Policy checking problems

The work of Li *et al.* [13] considers the interaction between *static* separation-of-duty constraints and resiliency policies. In this section, we reevaluate the problems they studied using the tools from fixed-parameter tractability, and extend the analysis to include user-independent constraints.

Before doing so, we note there are two types of constraints that can be defined. One type, which we will call *existential* constraints, are satisfied if some condition "guarded" by an existential quantifier holds. A resiliency policy $\mathsf{res}(P, s, d, t)$, for example, is an existential policy (since we need only find one allocation of users to teams). The other type, which we will call *universal* constraints, are satisfied if some condition "guarded" by a universal quantifier holds. A static separation-of-duty constraint $\mathsf{ssod}(Q, t)$ is a universal policy (since every subset of cardinality less than or equal to t must satisfy a particular condition).

A satisfiable instance of WSP indicates the existence of a plan. Informally, then, WSP can be used to determine the satisfaction (or otherwise) of an existential constraint (as we have seen with resiliency policies). Conversely, WSP can be used to determine the violation (or otherwise) of a universal constraint (as we have seen with static separation-of-duty policies).

Clearly, there is some "tension" between resiliency and SSoD policies: informally, resiliency policies require "over-provisioning" in the authorization policy, while SSoD policies require "minimal" provisioning. This leads naturally to the following definition and problem.

Definition 4. *The* Mixed Policy Checking Problem *is the problem of determining whether a set of resiliency and SSoD policies is satisfied by a given user-resource relation UR.*

Note the Mixed Policy Checking Problem is not considered as a problem in its own right in the work of Li *et al.* [13]. However, it turns out that an algorithm to solve the Mixed Policy Checking Problem can be used as a sub-routine to solve the Policy Consistency Checking Problem problem defined later. Now, the Mixed Policy Checking Problem can be solved by first asking whether every resiliency policy is satisfied (using Construction 1) and then asking whether any one of the static separation-of-duty policies is violated (using Construction 2). (We have to perform these checks separately because resiliency policies are existential constraints and separation-of-duty policies are universal.) Recall the input to MPCP includes a set F of resiliency and SSoD policies. We denote by s_M, d_M, t_M the maximum values of s, d, t, respectively, in any resiliency policy $\mathsf{res}(P, s, d, t) \in F$, and by p_M the size of the largest scope of any resiliency or SSoD policy of F. Given that testing the satisfiability of an SSoD policy $\mathsf{ssod}(P, t)$ is FPT parameterized by p, and that testing the satisfiability of a resiliency policy $\mathsf{res}(P, s, d, t)$ is FPT parameterized by (p, s, d, t), we obtain the following result:

Theorem 6. Mixed Policy Checking Problem *is FPT parameterized by (p_M, s_M, d_M, t_M).*

A set of policies comprising resiliency and SSoD policies is said to be *consistent* if and only if there exists an authorization relation UR such that every policy is satisfied [13]. This definition leads naturally to the following decision problem.

Definition 5. *The* Policy Consistency Checking Problem *consists in deciding whether there exists a user-resource relation UR that satisfies a given set of SSoD and resiliency policies.*

Li *et al.* [13] show that Policy Consistency Checking Problem is a computationally hard problem: it is in general in $\mathrm{NP}^{\mathrm{NP}}$, and the two restrictions where there is only one resiliency policy, and only one SSoD policy are NP-hard and coNP-hard, respectively. On the other hand, there exists a naive algorithm to solve the Policy Consistency Checking Problem using an algorithm for Mixed Policy Checking Problem as a sub-routine: first enumerate all possible authorization policies, and for each of them, construct an instance of Mixed Policy Checking Problem in order to test whether it satisfies the policies. The obvious question with this strategy is whether the algorithm will terminate, since the number of all possible authorization policies is, *a priori*, not bounded. In the following result, we show that this user set can actually be bounded by a function of the maximum size of the scopes of the input policies, implying an FPT algorithm with this parameter. Let $\mathsf{ssod}(P_1, t_1), \ldots, \mathsf{ssod}(P_{q_1}, t_{q_1})$ and $\mathsf{res}(P_{q_1+1}, 0, 1, t_{q_1+1}), \ldots, \mathsf{res}(P_{q_2}, 0, 1, t_{q_2})$ be the policies in the input of Policy Consistency Checking Problem (as Li *et al.* [13] note, we may suppose that $s = 0$ and $d = 1$ for all resiliency policies $\mathsf{res}(P, s, d, t)$). We note $P = \bigcup_{i=1}^{q_2} P_i$, and, as before, $p = |P|$.

Theorem 7. Policy Consistency Checking Problem *is FPT parameterized by p.*

Proof. First observe that for any set U and $UR \subseteq U \times R$ such that UR satisfies all policies, removing from UR all elements (u, r) such that $r \notin P$ does not violate any of the constraints. Then, we may also assume that for all $u \in U$, $N(u) \subseteq P$. Now we show that we may also delete every user u having a *twin*, *i.e.* a user u' such that $u \neq u'$ and $N(u) = N(u')$. Indeed, let u, u' be two twins, $U' = U \setminus \{u'\}$, and $UR' = UR|_{U'}$. First assume that UR satisfies two policies $\mathsf{ssod}(P, \ell)$ and $\mathsf{res}(P', 0, 1, t)$. Then, clearly UR' also satisfies $\mathsf{ssod}(P, \ell)$, since every subset V of U' of size at most $\ell - 1$ is also a subset of U of the same size, and thus $N(V) \subsetneq P$. Then, UR' satisfies $\mathsf{res}(P', 0, 1, t)$, since $N(V) = N((V \setminus \{u'\}) \cup \{u\})$ for all $V \subseteq U$ such that $u' \in V$.

Conversely, if UR' satisfies $\mathsf{ssod}(P, \ell)$ and $\mathsf{res}(P', 0, 1, t)$, then the same holds for UR, since, here again, $N(V) = N((V \setminus \{u'\}) \cup \{u\})$ for all $V \subseteq U$ such that $u' \in V$. Hence, we showed that for all U and $UR \subseteq U \times R$ such that UR satisfies all policies, we can iteratively delete all twins without violating any constraint, thus obtaining a solution U' and UR' with $|U'| \leq 2^p$. Hence, instead of enumerating all authorization policies, it is actually sufficient to enumerate all subsets of a user set $\mathcal{U}$ containing 2^p users, namely one user per subset of P, which represent the resources (s)he is authorized for. Then, for each such authorization policy, it remains to test the satisfaction of the separation-of-duty and resiliency policies (*e.g.*, using the algorithm of Theorems 3 and 5). One can observe that the total running time of this algorithm is $O^*(2^{p\log(p)+2^p})$. $\square$

4.2 Minimizing the number of needed users

The final problem that Li *et al.* [13] consider is the special case that the resiliency policy $\mathsf{res}(P, s, 1, \infty)$ and static separation-of-duty policy $\mathsf{ssod}(P, k, s)$ are defined over the same set of resources $P \subseteq R$. Given enough users, it will clearly be possible to find an authorization relation that simultaneously satisfies both policies. The interesting question is what the minimum number of users is. More formally, Li *et al.* define the following problem.

Definition 6. *Given a set of resources R and $\mathsf{res}(P, s, 1, \infty)$, $\mathsf{ssod}(P, k, s)$, with $P \subseteq R$, $s, k \in \mathbb{N}$,* Min-Users PCCP Satisfiability *(*MUPS*) consists of determining the smallest integer m_0 for which there exist U, $UR \subseteq U \times R$ such that $|U| \leq m_0$ and UR satisfies $\mathsf{res}(P, s, 1, \infty)$ and $\mathsf{ssod}(P, k, s)$.*

Li *et al.* [13] were not able to settle the complexity of Min-Users PCCP Satisfiability. However, they were able to obtain upper and lower bounds for m_0, these bounds being $p(s + 1)$ and $k + s$, respectively; moreover, they showed that the lower bound is tight whenever $p \geq \binom{k+s}{s+1}$.

While the general complexity of MUPS remains open, we can establish that it is FPT. In particular, the algorithm discussed in the proof of Theorem 7 can be modified to determine whether a set of resiliency and SSoD policies are satisfied by an authorization policy defined on a set of at most m users, where m is an additional input of the problem. Indeed, one just need to restrict the enumeration of all subsets of $\mathcal{U}$ to those of size at most m. We thus have the following result.

Theorem 8. MUPS *is FPT parameterized by p.*

5. CONCLUDING REMARKS

The specification and enforcement of policies and constraints is perhaps the most common way of implementing access control requirements in computer systems. These policies and constraints may be mutually incompatible to some extent, in the sense that authorization policies that satisfy one constraint may lead to the violation of another. In particular, static separation-of-duty and resiliency policies may be incompatible in this sense.

Hence, it is important to be able to analyze whether a given authorization policy simultaneously satisfies mutually incompatible constraints. Prior work in this area has established that such analyses are generally intractable [13]. In this paper, we make important contributions to the understanding of the Resiliency Checking Problem and the Policy Consistency Checking Problem from the perspective of fixed-parameter tractability.

First, we exhibit a transformation from the Resiliency Checking Problem to the Workflow Satisfiability Problem, by which a resiliency policy is translated into atmost and $\mathsf{atleast}$ constraints. This enables us to re-use and adapt known results for WSP, leading to new results establishing that Resiliency Checking Problem is FPT. In addition, the existence of optimized algorithms to solve WSP suggests that it will be possible to solve much larger instances of the Resiliency Checking Problem than has been possible up to now. Moreover, the fact that WSP is FPT for all user-independent constraints means we can introduce potentially useful generalization of and variations on the Resiliency Checking Problem. Second, we introduce the Mixed Policy Checking Problem, an intermediate problem that can be solved using methods similar to those for the Resiliency Checking Problem (ie, by translating to a WSP instance), and as a basis for solving the Policy Consistency Checking Problem. We show that the Mixed Policy Checking Problem is FPT and how an algorithm to solve the Mixed Policy Checking Problem can be used to solve the Policy Consistency Checking Problem. The method we use introduces a way of bounding the number of users that need to be considered, thus making the Policy Consistency Checking Problem FPT. This method may be of independent interest in the context of workflow satisfiability and is something we intend to explore in future work. We also note that we have recently extended the work in this paper to provide an exhaustive multi-variate analysis of the complexity of Resiliency Checking Problem [5].

It is perhaps worth mentioning in conclusion that WSP cannot be used directly to solve problems related to *workflow resiliency* [15]. In the case of resiliency in access control systems, the Resiliency Checking Problem asks whether it is possible to allocate users to teams even if up to s users are unavailable. Here the assumption seems to be that the set of (available) users will not change once the users have been allocated to teams. In the case of resiliency in workflow systems, however, three different models of user availability have been proposed [15], based on the assumption that workflow instances may run for a relatively long period of time, and users that were available when the workflow was instantiated may no longer be available when the workflow instance is partially complete. This leads to three distinct definitions of workflow resiliency: static, incremental and dynamic. The work on resiliency in access control essentially corresponds to static resiliency and questions of static resiliency can, presumably, be answered by reducing the problem to (multiple instances of) WSP. In future work, we plan to investigate incremental and dynamic workflow resiliency, using techniques for Valued WSP [4] (since we can use weights to model availability of users).

Finally, we note Khan and Fong's work on *workflow feasibility* [12] in the context of a constrained workflow authorization schema that also includes rules for updating the authorization relation UR. A workflow is feasible if there exists a "reachable" authorization relation (by application of the update rules) such that the resulting constrained workflow authorization schema is satisfiable. Khan and Fong's work was set in the specific context of relationship-based access control, where updates to the authorization relation are controlled in particular ways. Workflow feasibility is obviously related to the Policy Consistency Checking Problem and can presumably be modeled as multiple instances of WSP. In future work we hope to consider workflow feasibility in the context of role-based access control (RBAC) together with a suitable administrative model for RBAC to update the user-role and role-step relationships (from which the authorization relation may be derived).

Acknowledgements

This research was partially supported by EPSRC grant EP/K005162/1. Gregory Gutin's research was also supported by Royal Society Wolfson Research Merit Award.

6. REFERENCES

[1] American National Standards Institute. *ANSI INCITS 359-2004 for Role Based Access Control*, 2004.

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

[3] Cohen, D., Crampton, J., Gagarin, A., Gutin, G., and Jones, M. Iterative plan construction for the workflow satisfiability problem. *J. Artif. Intell. Res. (JAIR) 51* (2014), 555–577.

[4] Crampton, J., Gutin, G., and Karapetyan, D. Valued workflow satisfiability problem. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies, Vienna, Austria, June 1-3, 2015* (2015), E. R. Weippl, F. Kerschbaum, and A. J. Lee, Eds., ACM, pp. 3–13.

[5] Crampton, J., Gutin, G., and Watrigant, R. A multivariate approach for checking resiliency in access control. *CoRR 1604.01550* (2016).

[6] Cygan, M., Dell, H., Lokshtanov, D., Marx, D., Nederlof, J., Okamoto, Y., Paturi, R., Saurabh, S., and Wahlstrom, M. On problems as hard as CNF-SAT. In *Proceedings of the 2012 IEEE Conference on Computational Complexity (CCC)* (Washington, DC, USA, 2012), CCC '12, IEEE Computer Society, pp. 74–84.

[7] Downey, R. G., and Fellows, M. R. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.

[8] Fomin, F. V., Grandoni, F., and Kratsch, D. Measure and conquer: Domination - a case study. In *Automata, Languages and Programming*, L. Caires, G. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, Eds., vol. 3580 of *Lecture Notes in Computer Science.* Springer Berlin Heidelberg, 2005, pp. 191–203.

[9] Gutin, G., Kratsch, S., and Wahlström, M. Polynomial kernels and user reductions for the workflow satisfiability problem. In *Parameterized and Exact Computation - 9th International Symposium, IPEC 2014, Wroclaw, Poland, September 10-12, 2014. Revised Selected Papers* (2014), pp. 208–220.

[10] Impagliazzo, R., Paturi, R., and Zane, F. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci. 63*, 4 (2001), 512–530.

[11] Karapetyan, D., Gagarin, A. V., and Gutin, G. Pattern backtracking algorithm for the workflow satisfiability problem with user-independent constraints. In *Frontiers in Algorithmics - 9th International Workshop, FAW 2015, Guilin, China, July 3-5, 2015, Proceedings* (2015), J. Wang and C. Yap, Eds., vol. 9130 of *Lecture Notes in Computer Science*, Springer, pp. 138–149.

[12] Khan, A. A., and Fong, P. W. L. Satisfiability and feasibility in a relationship-based workflow authorization model. In *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings* (2012), S. Foresti, M. Yung,

and F. Martinelli, Eds., vol. 7459 of *Lecture Notes in Computer Science*, Springer, pp. 109–126.

[13] Li, N., Wang, Q., and Tripunitara, M. V. Resiliency policies in access control. *ACM Trans. Inf. Syst. Secur. 12*, 4 (2009).

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

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

APPENDIX

A. PROOFS

Proof of Theorem 2. Given an instance $(U, R, UR, \mathsf{res}(P, s, d, t))$ of RCP$\langle\rangle$, and $U' \subseteq U$, we define $UR|_{U'} = \{(u, r) \in UR : u \in U'\}$, the restriction of UR to users which belong to U'. By Definition 1, if UR does not satisfy $\mathsf{res}(P, s, d, t)$ then there is a set $V \subseteq U$ of cardinality s, called a *blocker set*, such that for any d mutually disjoint sets of users $V_1, \ldots, V_d$ such that $N(V_i) \supseteq P$ and $|V_i| \leqslant t$, $V_1 \cup \cdots \cup V_d \cap V \neq \emptyset$.

Now, consider the complement of RCP$\langle\rangle$, co-RCP$\langle\rangle$, where we wish to find a blocker set $V \subseteq U$ of size at most s (*i.e.* such that $UR|_{U \setminus V}$ does not satisfy $\mathsf{res}(P, 0, d, t)$). We design an algorithm for a more general version of this problem, which we call co-RCP with advice, where the input comes together with a set $V \subseteq U$, and the goal is to find a set $V^* \subseteq U$, such that $V^* \supseteq V$, $|V^*| \leq s$, and $UR|_{U \setminus V^*}$ does not satisfy $\mathsf{res}(P, 0, d, t)$. By the foregoing, solving co-RCP with advice with $V = \emptyset$ clearly solves RCP$\langle\rangle$. Recall that a negative answer for co-RCP with advice means that the RCP$\langle\rangle$ instance is positive, and conversely. In the following, we will only focus on solving co-RCP with advice. Let us denote by $\mathcal{A}$ the algorithm described in the statement.

We are thus given $(U, R, UR, \mathsf{res}(P, 0, d, t))$ and $V \subseteq U$. If $|V| > s$, then we can clearly answer *no* for co-RCP with advice. We thus assume in the following that $|V| \leq s$. Using algorithm $\mathcal{A}$, we determine whether $\mathsf{res}(P, 0, d, t)$ is satisfied by $UR|_{U \setminus V}$. In the negative case, then $V^* = V$ is a blocker set for co-RCP with advice, and we answer *yes*. Otherwise, $\mathcal{A}$ outputs a set of teams $V_1, \ldots, V_d$, with $V_i \subseteq U \setminus V$ for all $i \in [d]$. For all $u \in V_1 \cup \cdots \cup V_d$, we make a recursive call to our algorithm with input $(U, R, UR, \mathsf{res}(0, d, t))$ and $V \cup \{u\}$, and return *yes* if and only if one of these sub-instances returns *yes*. If the instance is a positive one for co-RCP with advice, then observe that any blocker set V^* must intersect $V_1 \cup \cdots \cup V_d$, and thus one of the recursive calls must return *yes*. If the instance is a negative one, then clearly all these recursive calls will return *no* as well, proving the correctness of the algorithm. Finally, observe that we branch on at most dt sub-instances of co-RCP with advice, each of them having its input V increased by one element. Since the algorithm stops whenever $|V| > s$, the number of calls to the algorithm is at most $O^*((dt)^s) = O^*(2^{s \log(dt)})$. Since in every execution we make at most one call to algorithm $\mathcal{A}$, the total running time of the algorithm is $O^*(2^{s \log(dt)} f(p, d, t))$. $\square$

Proof of Lemma 2. If W is satisfiable, then there exists a valid plan $\pi : P \to U$. In particular, $|\pi(P)| \leq t - 1$, which means that there exists a set of at most $t - 1$ users which, altogether, have access to all resources in P, thus violating the policy.

Conversely, assume that $\mathsf{ssod}(P, t)$ is violated, which means that there exists a set V of at most $t - 1$ users such that $N(V) \supseteq P$, *i.e.* for all $r \in P$, there exists $u_r \in V$ such that $(u_r, r) \in UR$. Thus, the plan $\pi : P \to U$ defined by $\pi(r) = u_r$ for all $r \in S$ is a valid plan for W. $\square$