Detecting Conflicts of Interest[*]

Paolo Giorgini, Fabio Massacci, John Mylopoulos, Nicola Zannone
University of Trento
name.surname@dit.unitn.it

Abstract

System vulnerabilities are often caused by the presence of conflicts within the organization where the system-to-be will eventually operate. In particular, conflicts of interest are very harmful since actors can exploit their positions/roles relative to the system for gaining personal advantage. Capturing and resolving such conflicts is a necessary condition for developing secure information systems.

In this paper, we show how conflicts of interest can be formally detected during requirements analysis. This allows system designers to investigate the causes for which conflicts may occur in an organization. Thereby, they can better understand the organizational structure and so provide appropriate countermeasures to resolve or at least mitigate them.

1 Introduction

Security has become a critical issue for software system development and many methodologies that incorporate security in the software engineering process exist [2, 9, 10, 13]. Most proposals deal with security in system-oriented terms: they focus on protection aspects through security services and related protection mechanisms.

Yet, security is often compromised by exploiting loopholes in a social-technical system as a whole, i.e., vulnerabilities in the combination of security mechanisms with other mechanisms or the human procedures integrating those mechanisms, rather than breaking security mechanisms.

A major source of system vulnerabilities is the presence of conflicts among requirements. This issue has spurred a number of researchers to classify conflicts and propose solutions to mitigate them [4, 6, 7, 11, 13].

[*]We thank Tobias Mahler for useful discussions on legal aspects of conflicts of interest. This work was partly supported by the projects RBNE0195K5 FIRB-ASTRO, 016004 IST-FP6-FET-IP-SENSORIA, 27587 IST-FP6-IP-SERENITY, 2003-S116-00018 PAT-MOSTRO, 1710SR-B/P PAT-STAMPS.

In this setting, conflicts of interest are critical in the development of secure systems since "trusted" actors often abuse their position to gain personal advantages. For example, in the early nineties John Rusnak gained nearly \$500.000 in bonuses for fake bank profits by exploiting his trader position at Allied Irish Bank to cover the loss of approximately \$700 millions in currency transactions [5, 8].

Unfortunately, current solutions for conflicts of interest are unsatisfactory for two reasons: firstly, proposed taxonomies of conflicts [7, 11, 13] are based on mechanisms for preventing them, instead of understanding why and when they occur; secondly, these solutions hardly make any reference to the legal theory on which they are based (see e.g. [12] for the Italian legal basis). Thus, their definitions are intuitively presented, but not justified. For example, van Lamsweerde et al. [13] define conflicts of interest as divergences between goals, Moffett et al. [6] as the overlap of the subjects of two authority policies, and Nyanchama et al. [7] as privilege-privilege conflicts and role-role conflicts in role-based access control.

Our objective is to support system designers in the detection of conflicts of interest during requirements analysis. This will aid system designers to adopt appropriate countermeasures to resolve or at least mitigate them. In our previous work [3], we introduced Secure Tropos, a formal framework for modeling and analyzing both functional and security requirements of the organizational environment of a system, as well as the system itself. Using this framework, we have dealt with conflicts among authorizations and obligations [3]. In this paper, we (1) investigate the sources of conflicts of interest at a requirements level grounding our notions on legal theory, and (2) sketch a clear-cut formalization that allows for automatic detection of conflicts.

In the next section we provide a description of the Secure Tropos methodology. Then, we extract the principles underlying conflicts of interest from legislation, and show how they can be captured through Secure Tropos models (§3). We present an axiomatization for detecting conflicts in a logic framework (§4). Next, we discuss technical issues (§5). Finally, we present related work and conclude with a summary of our contribution (§6).

2 Secure Tropos

Secure Tropos [3] uses the concepts of actor, goal, task, resource and social relationships for defining needs, entitlements and capabilities of actors, namely *objectives*, *ownership*, and *provisioning*. Objectives are goals intended to be attained, tasks intended to be performed, and resources requested by an actor. As an abbreviation, in the sequel we will simply refer to any of the above as *obtaining a service*. Ownership refers to the authority of an actor concerning the access and disposition of a service. Provisioning represents the capability of supplying a service.

Furthermore, Secure Tropos supports the notion of *trust*, and *delegation* to transfer entitlements and responsibilities from an actor to another. Delegation marks a formal passage of authority or responsibility in the domain matched by the issuance of an act called "power of attorney" with which the actor receiving the power of attorney (the delegatee) is "attorney in fact" for the actor giving the power (the delegator) [12, art. 176].

Trust is a relation between two actors, representing the expectation of one actor (the trustor) about the capabilities and behavior of the other (the trustee). Essentially, trust is the mental counterpart of delegation. Therefore, trust is normally necessary for delegation. However, there may be delegation without trust. As in [3], we introduce *monitoring* as surrogate of trust. The idea is that an actor (the monitor) is appointed by the delegator to monitor whether the delegatee will not misuse services and fulfill assigned obligations.

We have recognized the distinction between relations involving permission and execution (obligations) to be essential for modeling functional and security requirements at the same time [3]. Thus, we have distinguished *delegation of permission* from *delegation of execution*, and *trust of permission* from *trust of execution* [3]. Similarly, monitoring can be distinguished into permission and execution ones [3].

3 Conflicts of Interest

An *interest* is both a right to property or for the use of property and a desire of something done. Therefore, an interest concerns both permission and execution aspects.

A law dictionary[1] definition of *conflict of interest* is:

> *"a situation in which a person has a duty to more than one person or organization, but cannot do justice to the actual or potentially adverse interests of both parties. This includes when an individual's personal interests or concerns are inconsistent with the best for a customer, or when a public official's personal interests are contrary to his/her loyalty to public business."*

[1]http://dictionary.law.com

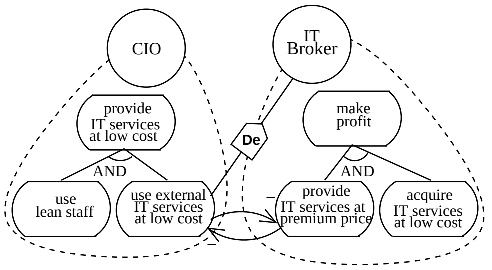
CIO
IT Broker
provide IT services at low cost
De
make profit
AND
AND
use lean staff
use external IT services at low cost
−
provide IT services at premium price
acquire IT services at low cost
−

Figure 1. Attorney-in-fact conflicts

In other words, a conflict of interest is a situation where an actor, such as a public official or a delegatee, has competing professional and/or personal interests that make it difficult to achieve his duties fairly.

Different forms of conflict of interest are often discussed along with access control models [7, 11]. In this setting, conflicts of interest are identified with conflicts of duties, i.e., failures of separation of duty constraints. However, this approach requires a priori knowledge of conflicts. On the contrary, our focus is on organizational requirements, and hence, on relations among actors. We classify conflicts of interest as follows:

- *Attorney-in-fact conflict*, where some (possibly personal) interests of the delegatee interfere with the interests of the delegator;
- *Role conflict*, where an agent is assigned a role whose interests collide with those of the agent;
- *Self-monitoring conflict*, where an actor is responsible for monitoring his own behavior.

While some conflicts can be captured by other methodologies (e.g. [7, 11, 13]), the attorney-in-fact conflict cannot. Due to space limitations, in the remainder of the paper we deal only with attorney-in-fact conflicts.

Tropos supports contribution analysis which allows designers to point out goals that can contribute positively or negatively to the fulfillment of a goal. We extend such an analysis in order to support the detection of conflicts of interest.

Consider the example in Figure 1. It presents a scenario which is common in IT procurements contracts of public institutions. The Chief Information Officer (CIO) is appointed by the company to provide IT services at low cost. In order to achieve this goal, he should use lean staff (in accordance to the Italian financial law) and use external IT services at low cost. The IT Broker has the objective to make profit that he achieves by acquiring IT services at low cost and providing IT services at premium price. The last goal negatively contributes to goal use external IT services at low cost aimed by the CIO (and vice versa). The CIO delegates the execution (a link labeled **De** in Fig. 1) of using external IT services at low cost to the IT Broker.

In this scenario, the IT Broker takes charge to achieve use external IT services at low cost. Clearly, this situation may lead to misdoing and, consequently, a possible harmful situation for the CIO. Actually, if the IT Broker achieves goal provide IT services at premium price, he cannot achieve the goal delegated by the CIO. On the contrary, if the IT Broker achieves the goal delegated by the CIO, he may penalize the agency where he works. Thereby, this situation represents a conflict of interest.

Unfortunately, Tropos alone cannot capture all cases because Tropos contribution analysis deals only with execution and so we need to extend it for dealing also with permission. If the fulfillment of a goal contributes or implies the fulfillment of another goal, this intuitively implies that implicit permission can be given. Moreover, Tropos conducts contribution analysis from the perspective of single actors. We must extend the analysis over the model representing the entire system and its environment for identifying conflicting interests among different actors.

Then, we have all necessary machinery to capture formally the notion of attorney-in-fact conflicts.

Definition 1 *An* attorney-in-fact conflict *occurs when*

1. *an actor (possibly directly) delegates the execution of a service whose fulfillment is denied by some objective of the delegatee;*
2. *an actor (possibly directly) delegates the permission on a service whose access is denied by some entitlement of the delegatee;*
3. *an actor (possibly directly) delegates the execution of a service that denies the fulfillment of some objective of the delegatee;*
4. *an actor (possibly directly) delegates the permission on a service that denies the access to some entitlement of the delegatee;*
5. *an actor (possibly directly) delegates the permission on a service that allows the delegatee to access another service for which the delegatee is not trusted by the legitimate owner.*

4 Automated Reasoning Support

Positive/negative contribution relations need to be specified manually in the requirements model. In simple examples (e.g., Fig. 1), such links are immediately visible, but in more complex scenarios that task might be difficult and error-prone. Therefore, as done in [3], we implement the semantics underlying Secure Tropos in the DLV system,[2] a Datalog solver. Datalog is a language of facts and rules. The collection of facts represents a Secure Tropos model. Table 1 extends the predicates presented in [3] introducing the ones used to detect attorney-in-fact conflicts.

[2]http://www.dbai.tuwien.ac.at/proj/dlv/

The graphical models of the system-to-be and its environment are insufficient for formal analysis [3]. Datalog supports the use of rules to complement the graphical notation. In addition to the rules presented in [3], we need rules for propagating entitlements and the fulfillment of services across contribution links. To this end, we use standard rules in contribution analysis.

Rules whose head is empty, are called *constraints*. In Datalog, constraints are used to specify conditions which must not be true in any model. In our setting, violations of constraints correspond to the presence of a conflict of interest in the system. Table 2 formalizes Definition 1.

Accordingly, the requirements analysis is composed of the following conceptual phases:

1. model the system;
2. complete the extensional description of the system;
3. verify the comprehensive description of the system.

In the modeling phase the designer draws the extensional requirements models where every node and edge correspond to a fact in the formal framework. Then, the reasoning system completes the extensional description of the system using rules and verifies its consistency using constraints. If any inconsistency is detected by the reasoning system, the designer needs to revise the requirements model in order to avoid or at least mitigate detected conflicts and repeats the formal analysis step.

5 Technical Issues

Unfortunately, contributions analysis alone is not sufficient to detect conflicts of interest. Actually, positive/negative contributions do not generate inconsistencies at functional/permission level as all explicit direct and indirect requirements are fulfilled. Therefore, we need to introduce explicitly the ability to model conflicts of interest. This requires to consider the relationships among actors and the roles an agent plays together with conflicting interests.

Our framework warns the designer on (all and only) the situations that *may* be harmful for some actor of the system. However, solutions for solving such conflicts depend on several factors such as cost, compliance with legislation and awkwardness of the conflict itself. Based on these factors, the designer can decide to adopt either no countermeasure, solutions for mitigating the conflicts or even modify the overall organizational structure of the system in order to eliminate the source of conflicts. Such a decision can be computer-aided through the use of ontologies, reference architectures and patterns supporting socio-technical system modeling.

$\mathsf{pos\_contribution}(\texttt{Service}: s_1, \texttt{Service}: s_2)$	s_1 positively contributes for delivering s_2 or having permission on s_2.
$\mathsf{neg\_contribution}(\texttt{Service}: s_1, \texttt{Service}: s_2)$	s_1 negatively contributes for delivering s_2 or having permission on s_2.
$\ni \texttt{Actor}: a\texttt{Service}: s$	a is the legitimate owner of s.
$\mathsf{has\_implicit\_per}(\texttt{Actor}: a, \texttt{Service}: s)$	a is implicitly authorized to access s. Implicit authorizations include both authorizations granted by the owner of the service and authorizations derived using contribution analysis.
$\mathsf{aims}(\texttt{Actor}: a, \texttt{Service}: s)$	a requests s or another actor has delegated him its execution.
$\mathsf{delegateChain}(perm, \texttt{Actor}: a, \texttt{Actor}: b, \texttt{Service}: s)$	a delegates (possibly directly) the permission on s to b.
$\mathsf{delegateChain}(exec, \texttt{Actor}: a, \texttt{Actor}: b, \texttt{Service}: s)$	a delegates (possibly directly) the execution of s to b.
$\mathsf{trustChain}(perm, \texttt{Actor}: a, \texttt{Actor}: b, \texttt{Service}: s)$	a trusts (possibly directly) that b will not misuse s.

Table 1. Predicates

A1	$\leftarrow \mathsf{delegateChain}(exec, A, B, S_1) \land \mathsf{neg\_contribution}(S_2, S_1) \land \mathsf{aims}(B, S_2)$
A2	$\leftarrow \mathsf{delegateChain}(perm, A, B, S_1) \land \mathsf{neg\_contribution}(S_2, S_1) \land \mathsf{has\_implicit\_per}(B, S_2)$
A3	$\leftarrow \mathsf{delegateChain}(exec, A, B, S_1) \land \mathsf{neg\_contribution}(S_1, S_2) \land \mathsf{aims}(B, S_2)$
A4	$\leftarrow \mathsf{delegateChain}(perm, A, B, S_1) \land \mathsf{neg\_contribution}(S_1, S_2) \land \mathsf{has\_implicit\_per}(B, S_2)$
A5	$\leftarrow \mathsf{delegateChain}(perm, A, B, S_1) \land \mathsf{pos\_contribution}(S_1, S_2) \land \ni CS_2 \land \mathsf{not\ trustChain}(perm, C, B, S_2)$

Table 2. Detecting Attorney-in-fact conflicts

6 Related Work and Conclusions

Conflicts of interest are often discussed in the context of role-based access control (RBAC) models. Several proposals attempt to integrate such models into Software Engineering by using or enhancing UML constructs [2, 9, 10]. In [2, 10], authors propose conceptual models for RBAC in UML where constraints are represented as classes. Ray et al. [9] integrate RBAC in UML as patterns using diagram templates, and represent separation of duty constraints in OCL. These approaches use specific domain constraints that will be checked statically or dynamically. However, they do not analyze organizational requirements to understand why such constraints should be introduced and the effects of their introduction, so major constraints could be omitted or minor constraints could affect system functionalities.

Other proposals provide mechanisms for the detection of conflicts. In [4, 6], authors propose to detect conflicts by analyzing overlaps in policies. Bandera et al. [1] use the Event Calculus to support this approach. They define constraints representing specific domain conflicts, and query the model for event sequences that prove the violation of such constraints. Another approach is proposed by van Lamsweerde et al. [13]. They argue that many requirements inconsistencies originate from conflicting goals and provide mechanisms for managing conflicts at a goal level. Our approach extends that work in that we consider both entitlements and objectives rather than only objectives.

This paper sketches the extension of the Secure Tropos formal framework in order to deal with conflicts of interest during requirements analysis. This framework with all new features presented in this paper is supported by the ST-Tool [3], a CASE tool for Secure Tropos. In particular, the tool is able to generate, from graphical models, Datalog specifications that are automatically verified by the DLV system. We leave details to the full paper.

References

[1] A. K. Bandara, E. Lupu, and A. Russo. Using Event Calculus to Formalise Policy Specification and Analysis. In *Proc. of POLICY'03*, pages 26–39. IEEE Press, 2003.

[2] D. Basin, J. Doser, and T. Lodderstedt. Model Driven Security: from UML Models to Access Control Infrastructures. *TOSEM*, 15(1):39–91, 2006.

[3] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modeling Security Requirements Through Ownership, Permission and Delegation. In *Proc. of RE'05*, pages 167–176. IEEE Press, 2005.

[4] E. C. Lupu and M. Sloman. Conflicts in Policy-Based Distributed Systems Management. *TSE*, 25(6):852–869, 1999.

[5] F. Massacci and N. Zannone. Detecting Conflicts between Functional and Security Requirements with Secure Tropos: John Rusnak and the Allied Irish Bank. Technical Report DIT-06-002, University of Trento, 2006.

[6] J. D. Moffett and M. S. Sloman. Policy Conflict Analysis in Distributed System Management. *J. of Organisational Comp.*, 4(1):1–22, 1994.

[7] M. Nyanchama and S. Osborn. The role graph model and conflict of interest. *TISSEC*, 2(1):3–33, 1999.

[8] Promontory Financial Group, Wachtell, Lipton, Rosen, and Katz. Report to the Board and Directors of Allied Irish Bank P.L.C., Allfirst Financial Inc., and Allfirst Bank Concerning Currency Trading Losses, March 12, 2003.

[9] I. Ray, N. Li, R. France, and D.-K. Kim. Using UML to visualize role-based access control constraints. In *Proc. of SACMAT'04*, pages 115–124. ACM Press, 2004.

[10] M. E. Shin and G.-J. Ahn. UML-Based Representation of Role-Based Access Control. In *Proc. of WETICE'00*, pages 195–200. IEEE Press, 2000.

[11] R. Simon and M. E. Zurko. Separation of duty in role-based environments. In *Proc. of CSFW'97*, pages 183–194. 1997.

[12] P. Trimarchi. *Istituzioni di diritto privato*. Giuffrè Editore, XVI edition, 2005.

[13] A. van Lamsweerde, R. Darimont, and E. Letier. Managing Conflicts in Goal-Driven Requirements Engineering. *TSE*, 24(11):908–926, 1998.