Computer Aided Threat Identification

Yudistira Asnar
DISI-University of Trento, Italy
yudistira.asnar@unitn.it

Tong Li
School of Software, Tsinghua University, China
litong30@gmail.com

Fabio Massacci
DISI-University of Trento, Italy
fabio.massacci@unitn.it

Federica Paci
DISI-University of Trento, Italy
federica.paci@unitn.it

***Abstract*—Recently, there has been an increase of reported security threats hitting organizations. Some of them are originated from the assignments to users of inappropriate permissions on organizational sensitive data. Thus it is crucial for organizations to recognize as early as possible the risks deriving by inappropriate access right management and to identify the solutions that they need to prevent such risks.**

In this paper, we propose a framework to identify threats during the requirements analysis of organizations' IT systems. With respect to other works which have attempted to include security analysis into requirement engineering process (e.g., KAOS, Elahi et al., Asnar et al.), our framework does not rely on the level of expertise of the security analyst to detect threats but allows to automatically identify threats that derive from inappropriate access management.

To capture the organization's setting and the system stakeholders' requirements, we adopt SI* [1], a requirement engineering framework founded on the concepts of *actors, goals, tasks* and *resources*. This framework extends SI* with a reasoning technique that identifies potential security threats on resources and relevant goals. The reasoning is based on Answer Set Programming (ASP) logic rules that take into account the relationships between resources and the delegation of permission relations between actors. We illustrate this framework using an eHealth scenario.

I. Introduction

The intricacies of controls and permission assignments become more complex as the companies' size, the number of systems, and the granularity of access rights within enterprise-spanning systems grow. In organizations, managing the access rights while protecting sensitive organizational information is a daunting process for security administrators. Many users may not be aware of their access rights or conflicts/issues that their rights could create. Often users are granted access to sensitive information but they are not trusted by the information's owners. If users misuse their permissions they can un/intentionally compromise organizational assets' security. Examples of threats that can hit an organization because of an inappropriate access management are unintended data editing, un/intentional disclosure, or internal misuse for a personal gain.

To avoid such kind of threats, it is crucial to identify pitfalls in users' permissions assignments in the early phases of an information system engineering process and manage the level of risk of such threats by adopting suitable security solutions.

In this paper, we propose a framework to identify threats that are caused by inappropriate permissions assignment. We identify such threats during the requirement analysis of the organizations settings and of its IT systems. With respect to other works which have attempted to include security analysis into requirement analysis ([2]–[5]), our framework does not rely on the level of expertise of the security analyst to detect threats but allows to automatically identify threats that derive from inappropriate access management.

To capture the organization's setting and the system stakeholders' requirements, we adopt SI* [1], a requirement engineering framework founded on the concepts of *actors*, *goals*, *tasks* and *resources*: *actors* are the system's stakeholders, *goals* are actors' requirements, and *tasks* and *resources* are the means to achieve actor's goals.

In this work, we consider resources and goals as assets because their fulfillment brings value to the organization. Therefore, they must be protected by assuring that security properties (e.g., confidentiality, integrity, and availability) are not violated.

We have extended SI* with a reasoning technique that identifies potential security threats on resources and relevant goals. A threat is a situation where an actor who is not trusted by the owner of the goal/resource is granted a permission on a goal/resource that the actor might misuse to harm the goal/resource. To enable the reasoning, we extend the SI* modeling language with different types of relationships between resources and different permission types on resources.

The reasoning is thus based on Answer Set Programming (ASP) logic rules that take into account the relationships between resources and the delegation of permission relations between actors.

The rest of the paper is organized as follows. In Section II, we introduce the running example taken from the healthcare domain. We introduce SI* framework in Section III. We present the extensions to the SI* in Section IV and the reasoning process to identify threats in Section V. Then, we

discuss the related work in Section VI and outline future work in Section VII.

II. Running Example - Drugs Management

To illustrate our approach for identifying threats, we use a scenario taken from the healthcare domain that is about health monitoring and drugs delivery to patients' home. This example involves six main actors:

- **Patient** is monitored by a smart T-shirt which measures medical data (e.g., heartbeat rate, blood pressure, etc.) and transfers them to the Hospital's computer system. When the patient's condition is abnormal, the doctor makes a diagnosis and produces a prescription. The patient receives his prescription and requests the drug delivery service to the pharmacy.
- **Hospital** provides medical services to patients. The hospital monitors patients' health and manages patients' data, which are stored in the hospital's computer. When the patient has some problems, the hospital assigns a doctor to diagnose the patient.
- **Pharmacy** is responsible for managing drugs and provide them to the patients. All the information about drugs is stored in the pharmacy's computer.
- **Pharmacist** works for the pharmacy and is responsible to provide drugs to be delivered according to the prescription received from the patient. The prescription information is stored in the pharmacy's computer.
- **Drug manager** works for the pharmacy and is responsible to manage the drugs. All the drugs' information is also stored in the pharmacy's computer.
- **Doctor** is responsible to diagnose the patients and prescribe required medications.

III. The SI* Modeling Framework

SI* [1], [5] is a modeling framework extending the i* framework [6] which supports security requirement analysis. SI* is part of a complete security methodology, which aims at analyzing and modeling organizational settings and its security and dependability requirements.

The SI* language [1] is founded on the concepts of *actor*, *goal*, *resource* and relations such as *AND/OR decomposition* and *means-end*. An actor is an entity which has intentions, capabilities, and entitlements; a *goal* captures a strategic interest that is intended to be fulfilled; a *resource* is an artifact produced/consumed by a goal; *AND/OR decomposition* is used to refine a goal; *means-end* identifies goals that provide means for achieving another goal or resources produced or consumed by a goal/task.

Example III.1. Figure 1[2] shows the goal model for the Drugs Management scenario involving six actors: the Hospital, the Doctor, the Patient, the Pharmacy, the Pharmacist, and the Drug Manager. The Hospital (*Actor*) has an intention (**R***equest*) to fulfill the goal Provide medical service and **O***wns* the Smart T-shirt resource, while the Doctor has the capability (**P***rovide*) to fulfill the goal Diagnose. This goal is then refined (*AND-decomposed*) into subgoals Monitor patient, Manage patient data, and Diagnose. Some goals can produce or consume some resources. For example, the goal Diagnose requires the resource Patient data and produces the resource Prescription.

SI* also captures social relationships (e.g., *delegation* and *trust*) for defining the entitlements, capabilities and objectives of actors. Originally, a *delegation* marks a formal passage of responsibility (*delegation execution*) or authority (*delegation permission*) from an actor (*delegator*) to the actor receiving the responsibility/authority (*delegatee*) to achieve a goal or to provide a resource. In this work, we extend the notion of *delegation of permission* as described in the following section. *Trust* is a relation between two actors representing the expectation of one actor (*trustor*) about the capabilities of the other (*trustee*) – *trust execution*, and about the behavior of the trustee with respect to the given permission – *trust permission*. In this work, we only consider the notion of *trust of permission*.

Example III.2. The Hospital *delegates execution* of the goal Diagnose to the Doctor, because the doctor has the capability to fulfill the goal. Moreover, the Patient believes that the Hospital will not misuse the permission given upon the Patient Data as specified by the *trust permission* relation between the Patient and the Hospital.

IV. The Extended Goal Model

In order to automatically detect threats on resources we rely on a subset of the basic concepts of SI* [1] and we propose some extensions to the SI* modeling language to represent different types of actors' permissions on resources and different types of relationships between resources.

We inherit from SI* the concepts of *actor*, *resource*, and *goal* and a subset of SI* relations such as *decomposition*, *means-end*, *provide*, *request*, *own*, and the social relations *delegation* and *trust* of permissions and execution that have been introduced in Section III.

In this work, goals and resources are considered as assets that need to be protected because they bring values to organizations. In order to specify how an asset needs to be protected, we use the concept of *security requirement* defining a specific security property, such as confidentiality, integrity, and availability, to be preserved for a resource. Moreover, a security requirement might also be related to a goal because this goal produces or consumes the resource.

Figure 2 illustrates the extended SI* conceptual model used in this work where in gray we indicate concepts and relations that belong to SI* and in black we denote our

[1] We only mention concepts that are relevant to this work

[2] We use *italic fonts* to indicate the basic concepts of SI* framework, and sans-serif fonts to denote the one related to the running example.

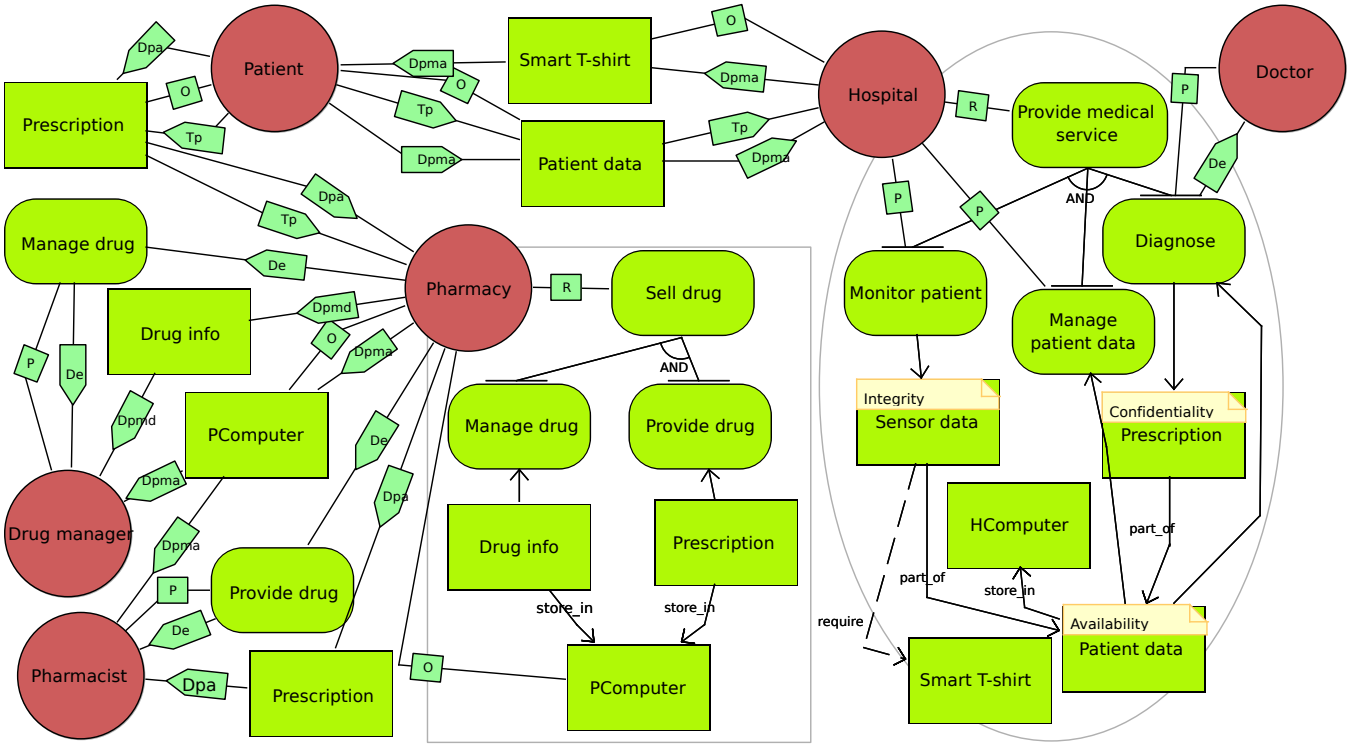

Figure 1. Goal Model of Drugs Management Scenario

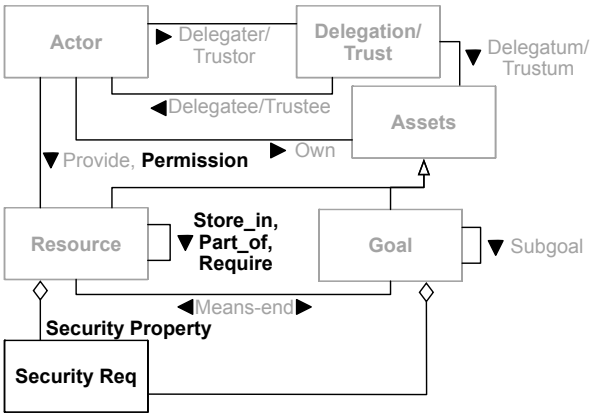

Figure 2. Conceptual model of the extended goal modeling framework

extensions. We introduce our extensions to SI* in the next two subsections.

A. Types of Resource Permissions

SI* framework can only capture whether an actor has a permission on a resource or not but it does not allow one to specify which is the type of permission the actor is granted on the resource. Specifying the permission type is crucial because it determines the type of actions an actor can perform on a resource. Some of these actions might be used by an actor to un/intentionally harm a resource.

Example IV.1. Considering the scenario in Figure 1, the Drug Manager needs the modify permission on Drug info to fulfill its goal Manage drug. However, it can be the case that the Drug Manager is given an higher permission (e.g., manage) on Drug info which gives him also the capability to revoke the permission given to other actors on Drug info and thus to make no longer available the resource to these actors. This situation might be a problem if the actors need to access Drug info to fulfill their goals.

Therefore, we refine the notion of permission in SI* by introducing three different types of permissions: *access*, *modify*, and *manage* as described in Table I.

Table I
PERMISSIONS ON RESOURCE

Permission Type	Description	(Possible) Affected Sec. Property
Access (low-level)	Actor only has the permission to access/read/use the resource.	Confidentiality
Modify (medium-level)	Actor can change the content of the resource.	Integrity
Manage (high-level)	Actor has the permission to modify the resource, delegate permissions to other actors and modify permissions to other actors.	Availability

Each permission type determines the set of actions that an actor can perform on a resource. Thus, a permission type

might lead to the violation of a specific security property if the actor misuses the actions associated with the permission type. In Table I we show the relation between permission types and security properties that might be violated if an actor abuses of the permission type. For example, if an actor has an *access* permission on a resource, he can accidentally disclose the resource and thus violate the resource's confidentiality.

Example IV.2. In Example IV.1, the Drug Manager has a *modify* permission on Drug info. If Drug Manager misuses this permissions he/she un/intentionally compromise the integrity of Drug info.

We also assume that there is a hierarchy between the different permission types: the *manage* permission implies the *modify* one, and the *modify* implies the *access*, because in an information system the *manage* permission is considered the most powerful right.

Example IV.3. The Drug manager has *modify* permission on Drug info resource therefore s/he can also *access*/read the Drug info.

Note that if a resource is a physical resource (e.g., DBMS, computer), we consider only *access* and *manage* permission because for such kind of resource it does not make sense to distinguish *modify* and *manage* permissions; while on an informational resource we might grant all three permission types.

Since an actor is able to delegate a permission to another actor, we need to extend the *delegation of permission* in order to specify which permission type is delegated. The delegation of the different types of permissions - delegation of access/modify/manage- is graphically represented by an edge labeled with **Dpa**, **Dpmd** and **Dpma** respectively as depicted in Figure 1. Note that, a delegator can only delegate a permission on a resource when s/he has the *manage* permission on the resource.

B. Relationships between Resources

The other important extension we propose to SI* modeling language is the ability to specify relations between resources. Essentially, resources in an information system can be structured using three relations: $store\_in$, $part\_of$, and $require$.

The $store\_in$ relation captures a situation where an informational resource is stored in a physical resource such as a file stored on a computer. By using this relation, we can determine all locations where a resource is stored and which actors have access to those locations; and whether a particular security property of the resource might be violated.

Example IV.4. Patients' prescriptions are not only *stored in* the Hospital's computer system, but also *stored in* the Pharmacy's computer. This means that the actors who have the permission to manage the Pharmacy's computer can also access/modify/manage the Prescriptions.

The $part\_of$ relation indicates a composition relation between resources. If an actor has some permission on a resource, it implies s/he has the same permission on its parts.

The $require$ relation denotes that a resource might require another resource to function. In contrast with $part\_of$ and $store\_in$, $require$ does not need that the same permissions apply to both resources linked by the relation.

Moreover, all the relations require the same security requirement is preserved for the linked resources.

Example IV.5. In order to guarantee the availability of the Sensor data also the availability of the Smart T-shirt needs to preserved. Similarly, in Example **??** to preserve the confidentiality of Patient data we need to guarantee the confidentiality of Prescription and Sensor data.

V. Threat Identification

The threat identification reasoning process aims at identifying potential security threats for resources in a goal model. We consider as a *threat* a situation where an actor has a particular permission on a resource/goal but the owner of the resource/goal does not trust the actor.

The reasoning process is based on ASP logic rules for i) formalizing the goal model (Tables II and III), ii) determining the actors' permission on resources (Table IV), iii) determining the security requirements that are associated with resources and goals (Table V), and iv) identifying threats (Table VI).

A. Build the Goal Model

The aim of this step is to draw a SI* model that captures the system's stakeholders requirements as depicted in Figure 1. This means identifying the actors with their goals and resources and tasks to fulfill them, and the delegation and trust relationships between actors. Moreover, it is important to identify the relationships between resources and resources' security requirements.

In order to identify the threats automatically, the SI* model has to be formalized. As we have done in [7], we use an ASP system (i.e., DLV) to formalize the new concepts and relations introduced in this work. Table II lists the predicates to formalize a SI* model similar with the one proposed in [1]. There are new predicates to express the relations between resources, the permission an actor has on a resource, the extended delegation of permission relation, the security requirements of resources and goals, and the threats on resources and goals.

The predicates *stored_in*, *part_of* and *require* denote the possible relationships between resources. The predicate *permission* represents the permission an actor has on a resource where $PT \in \{access, modify, manage\}$. An actor can

delegate its permission (*del_perm*) to another actor when he has sufficient permission on the resource (i.e., $manage$). Moreover, the predicate *trust_perm* indicates the belief of an actor that the other actor will not misuse the granted permission.

The predicate *secure_req* represents the security requirement associated with resources and goals. This predicate holds when a resource needs to meet a specific security property ($SP \in \{confidentiality, integrity, availability\}$), or a goal needs to meet a security property which holds for a specific resource that is produced or required by the goal. Finally, the predicate *threat* holds if an actor is a threat agent.

Example V.1. The SI* model, presented in Figure 1, is formalized as depicted in the snip of the DLV input in Figure 3. Lines 1 to 17 show the predicates to denote the actors, goals and resources in the SI* model. The remaining lines show the predicates that encode relations between actors, goals and resources. For example, lines 39 represents the delegation of execution of goal Manage Drug between the Pharmacy and the Drug Manager. Line 42 shows the delegation of *manage* permission on PComputer from the Pharmacy to the Drug Manager.

We also introduce axioms (Table III) to derive implicit relationships between resources and actors in the model. Axiom $A1$ specifies that if an actor is the owner of a resource R, he also owns each resource subpart R_1. Axiom $A2$ specifies that if an actor has the capability to provide a goal G, he is also able to provide G's subgoals. In this work, we assume the *trust* relationship is transitive across actors (Axiom $A3$) or via *part-of* relation (Axiom $A4$).

Example V.2. The Patient is the owner of the Patient data therefore s/he is also the owner of resource's subparts: Sensor data and Prescription as stated in Axiom $A1$. Assuming the Hospital has the capability to provide the goal Provide medical service, according to Axiom $A2$, then the Hospital also has capabilities to fulfill Provide medical service's subgoals: Monitor patient, Manage patient data, and Diagnose. According to axiom $A3$, the Patient believes the Doctor will not misuse any given permission on the Patient data because the Patient trusts the Hospital and the Hospital trusts the Doctor.[3] Since the Patient trusts the Hospital upon the *manage* permission to his/her Patient data, then indirectly s/he also trusts the Hospital to manage Patient data's subparts: Sensor data and Prescription(axiom $S4$).

Table II
PREDICATES FOR AN EXTENDED GOAL MODEL FORMALIZATION

Goal model
goal(Goal:g)
resource(Resource:r)
actor(Actor:a)
provide(Actor:a, Goal:g)
own(Actor:a, Goal:g)
own(Actor:a, Resource:r)
subgoal(Goal g1, Goal:g)
means_ end(Resource:r, Goal:g)
means_ end(Goal:g, Resource:r)
Resource model
stored_ in(Resource:r, Resource:r1)
part_ of(Resource:r, Resource:r1)
require(Resource:r, Resource:r1)
Permission model
permission(Actor:a, Resource:r, PType:pt)
del_ perm(Actor:a, Actor:a1, Resource:r, PType:pt)
trust_ perm(Actor:a, Actor:a1, Resource:r)
Security requirements and Threats model
secure_ req(Resource:r, SProperty:sp)
secure_ req(Goal:g, SProperty:sp, Resource:r)
threat(Actor:a, Resource:r, SProperty:sp)
threat(Actor:a, Goal:g, SProperty:sp, Resource:r)

Table III
AXIOMS FOR GOAL MODEL EXTENSIONS

Relation implication	
A1	$own(A, R1) \leftarrow part\_of(R1, R) \wedge own(A, R)$
A2	$provide(A, S1) \leftarrow subgoal(S1, S) \wedge provide(A, S)$
A3	$trust\_perm(A, A2, R) \leftarrow trust\_perm(A, A1, R) \wedge trust\_perm(A1, A2, R)$
A4	$trust\_perm(A, A1, R1) \leftarrow part\_of(R1, R) \wedge trust\_perm(A, A1, R)$

Table IV
RULES FOR ACTORS' PERMISSIONS DETERMINATION

Permission propagation	
P1	$permission(A, R, modify) \leftarrow permission(A, R, manage)$
P2	$permission(A, R, access) \leftarrow permission(A, R, modify)$
P3	$permission(A, R, manage) \leftarrow own(A, R)$
P4	$permission(A1, R, PT) \leftarrow del\_perm(A, A1, R, PT) \wedge permission(A, R, manage)$
P5	$permission(A, R1, manage) \leftarrow store\_in(R1, R) \wedge permission(A, R, manage)$
P6	$permission(A, R1, PT) \leftarrow part\_of(R1, R) \wedge permission(A, R, PT)$

Note that the quality of a goal model is crucial for the validity of the analysis's results. In [8] the authors have illustrated several best practices to build a "good" goal model.

B. Determine Actor's Permission on Resources

By applying the automated reasoning formalized in Table IV, we can determine the actors' permissions on each resource. Axioms $P1$ and $P2$ represent the hierarchy of permissions. Axiom $P3$ assume the owner of a resource will have the highest permission on the resource (i.e., *manage*). Axiom $P4$ specifies that an actor (with the *manage* permission on a resource) can give any permission type on the resource to another actor. Additionally, other actors' permissions can be derived from the relationships between resources. Axiom $P5$ specifies that if an actor has a *manage* permission on a resource R which stores a resource R_1, s/he then has the *manage* permission also on R_1. Axiom

[3]We are aware that in some domain the transitivity of trust relations might not be applicable.

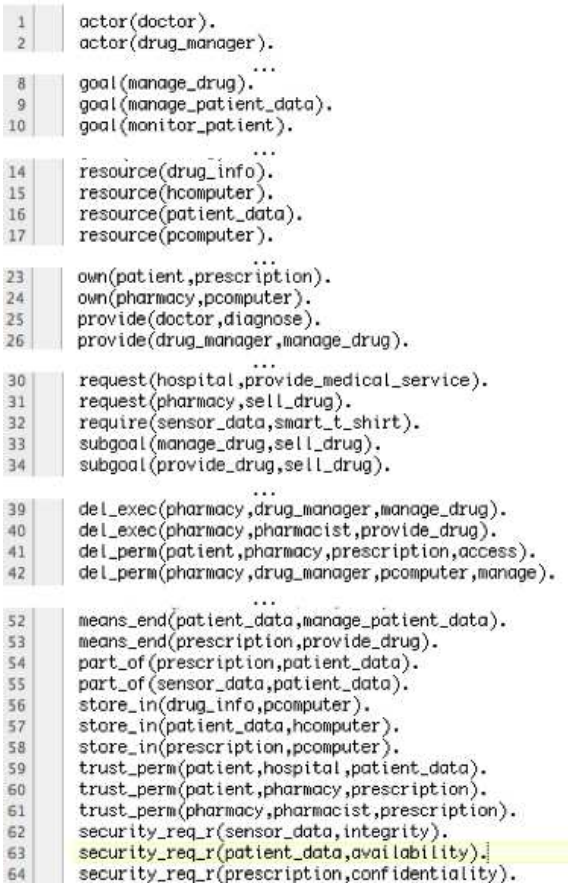

Figure 3. Drugs Management Goal Model Formalization

$P6$ specifies that if an actor has a permission PT on a resource R, then s/he has the same permission on subpart R_1 of resource R.

Example V.3. Since the Patient has delegated the *access* permission to the Pharmacy, the Pharmacy has the *access* permission on Prescription. Moreover, the Pharmacy has the *manage* permission on PComputer following axiom $P3$, and the *manage* permission on Drug Info and Prescription based on axiom $P5$. By applying axiom $P4$ the Pharmacist and the Drug Manager obtains the *manage* permission on the PComputer from the Pharmacy. Apart from this, the Pharmacist also gains *access* permission on the Prescription from the Pharmacy. Since the Drug Manager has *manage* permission on the PComputer (axioms $P1$ and $P2$) and Prescription is stored in PComputer,Drug Manager has *manage* permission on Prescription (axioms $P5$).

C. Determine Goals and Resources Security Requirements

We assume that if a security requirement has to be fulfilled for the resource R_1 then the same security requirement should be fulfilled for another resource R that is linked to R_1 by a relationship $store\_in$, $part\_of$, and $require$. The same assumption applies to a goal G which is linked to the resource by $means-end$ relation. Table V specifies the axioms to determine the security requirement for resources which are linked via a $store\_in$, $part\_of$, and $require$ relationship (axiom $S1$-$S7$). Axiom $S3$ states that if the integrity of a resource R_1 is critical and R_1 is stored inside another resource R, than also the integrity of R should be guaranteed. The security requirement of a goal can be derived from the resource's security requirement to which the goal is connected via a means-end relationship as specified by axioms $S8$ and $S9$. Axiom $S10$ specifies that the same security requirement that applies to a goal G should be applied to its subgoals.

Example V.4. Since the confidentiality of Prescription is required and Prescription is stored in PComputer because of Axiom $S1$ and $S2$ the confidentiality and integrity of PComputer has to be preserved as well. By applying axiom $S4$, in order to guarantee the availability of Patient Data, HComputer must be available. Moreover, since Sensor Data requires the Smart T-shirt and integrity of Sensor Data has to be preserved, the integrity of Smart T-shirt has to be guaranteed (axiom $S6$).

D. Identify Potential Threats

Table VI lists all the axioms used for identifying potential threats.[4] Essentially, once the security requirement of resources and goals and actors' permissions are defined, we can determine possible threats on resources following axioms $T1$-$T3$. The axioms state that we have a threat when an actor has a permission sufficient to violate the security requirement associated with the resource and the actor is not trusted by the resource's owner. Then, the threat for goals are identified by applying axioms $T4$-$T7$. A goal G is threatened when the goal is provided by an actor A_1 that is a threat agent for a resource R that is linked to the goal by a $means-end$ relation. For each threat, we also identify the threat agent, the resource being harmed and the security property being violated.

Example V.5. Table VII summarizes the threats identified. Drug Manager is a threat agent for both Prescription and PComputer because he has *manage* permission on PComputer and PComputer stores Prescription but the Patient does not trust the Drug Manager.

The list of threats can be useful for different stakeholders in the security engineering process. The risk analyst, for example, can assess the level of risk associated with the threats: if the risk is unacceptable, some treatments are required to mitigate the risk level. These treatments can be returned back to the requirement analyst who revises the SI* model with tasks representing the treatments. Alternatively,

[4] The complexity of identifying threats is the same as the one of checking a DLV program [9] because the model and the axioms are formalized as a DLV program.

Table V
AXIOMS FOR DERIVING SECURITY REQUIREMENTS

Security requirement propagation	
S1	$secure\_req(R, confidentiality) \leftarrow store\_in(R1, R) \wedge secure\_req(R1, confidentiality)$
S2	$secure\_req(R, integrity) \leftarrow store\_in(R1, R) \wedge secure\_req(R1, confidentiality)$
S3	$secure\_req(R, integrity) \leftarrow store\_in(R1, R) \wedge secure\_req(R1, integrity)$
S4	$secure\_req(R, availability) \leftarrow store\_in(R1, R) \wedge secure\_req(R1, availability)$
S5	$secure\_req(R1, SP) \leftarrow part\_of(R1, R) \wedge secure\_req(R, SP)$
S6	$secure\_req(R, integrity) \leftarrow require(R1, R) \wedge secure\_req(R1, integrity)$
S7	$secure\_req(R, availability) \leftarrow require(R1, R) \wedge secure\_req(R1, availability)$
S8	$secure\_req(G, SP, R) \leftarrow secure\_req(R, SP) \wedge means\_end(G, R)$
S9	$secure\_req(G, confidentiality, R) \leftarrow secure\_req(R, confidentiality) \wedge means\_end(R, G)$
S10	$secure\_req(G1, SP, R) \leftarrow subgoal(G1, G) \wedge secure\_req(G, SP, R)$

Table VI
AXIOMS FOR THREAT IDENTIFICATION

Threat Identification	
T1	$threat(A1, R, confidentiality) \leftarrow own(A, R) \wedge secure\_req(R, confidentiality) \wedge permission(A1, R, access) \wedge not\ trust\_perm(A, A1, R) \wedge A1 \neq A$
T2	$threat(A1, R, integrity) \leftarrow own(A, R) \wedge secure\_req(R, integrity) \wedge permission(A1, R, modify) \wedge not\ trust\_perm(A, A1, R) \wedge del\_perm(A, A1, R, PT) \wedge A1 \neq A$
T3	$threat(A1, R, availability) \leftarrow own(A, R) \wedge secure\_req(R, availability) \wedge permission(A1, R, manage) \wedge not\ trust\_perm(A, A1, R) \wedge A1 \neq A$
T4	$threat(A1, G, confidentiality, R) \leftarrow threat(A1, R, confidentiality) \wedge means\_end(R, G) \wedge provide(A1, G)$
T5	$threat(A1, G, confidentiality, R) \leftarrow threat(A1, R, confidentiality) \wedge means\_end(G, R) \wedge provide(A1, G)$
T6	$threat(A1, G, integrity, R) \leftarrow threat(A1, R, integrity) \wedge means\_end(G, R) \wedge provide(A1, G)$
T7	$threat(A1, G, availability, R) \leftarrow threat(A1, R, availability) \wedge means\_end(G, R) \wedge provide(A1, G)$

Table VII
IDENTIFIED THREATS

$threat(drug\_manager, prescription, confidentiality)$ $threat(drug\_manager, prescription, availability)$
$threat(drug\_manager, pcomputer, confidentiality)$ $threat(drug\_manager, pcomputer, integrity)$ $threat(drug\_manager, pcomputer, availability)$
$threat(pharmacist, prescription, confidentiality)$ $threat(pharmacist, prescription, availability)$
$threat(pharmacist, pcomputer, confidentiality)$ $threat(pharmacist, pcomputer, integrity)$ $threat(pharmacist, pcomputer, availability)$
$threat(pharmacist, provide\_drug, confidentiality, prescription)$

the threat list can be piped to the security architect who identify possible security solutions at architectural level to prevent such threats to occur. An example of how to to transform security requirements into architecture solution is presented in [10].

VI. RELATED WORK

In this work, we consider several requirement frameworks that have attempted to include security analysis into the requirement elicitation process. Among goal-oriented approaches, van Lamsweerde extends KAOS by introducing the notions of obstacle [11] and anti-goal [2] to analyze the security concerns of a system. KAOS obstacle captures an undesired state of affairs that might harm safety goals (i.e., hazard) or threaten security goals (i.e., threat). In the framework, the authors propose a formal framework to identify the obstacles to a goal in a given domain properties and to generate resolutions to those obstacles. KAOS anti-goal captures the intention of an attacker or considers it as a malicious obstacle. In comparison to ours, the security analysis proposed in these works does not take in to account actors' permissions and relations in the organization which can be considered as the main sources of security risks in an organization. Moreover, there is lack of guidance on how to specify domain properties so that the obstacles/anti-goals elicitation is complete and relevant.

Liu et al. [3] proposes an extension of the i* framework [6] to identify attackers, and analyze vulnerabilities through actor's dependency links. In this framework, all actors are considered as potential attackers therefore their capabilities are analyzed and possible damages caused by actors are assessed. In Li et al. [12], the authors proposed a formal framework to support the attacker analysis. Similarly, Elahi et al. [4] propose extensions to i* to model and analyze the vulnerabilities affecting systems requirements. Though these works consider relations between actors, they still rely on the level of expertise of the security analysts to identify possible attackers (i.e., even they have already been part of the model) and vulnerability. Moreover, the notion of permission (including the "delegation of permission") is not considered to be critical in their modeling and analysis while it is crucial in our framework.

In addition, some works focus on integrating risk analysis into the requirement analysis process, such as ISSRM [13], the GR framework [5], and CORAS [14]. In Information

System Security Risk Management, Mayer et al. [13] proposed a conceptual model for managing security of an information system based on several security methods (e.g., CORAS, ISO 27001). Unlike ISSRM, Asnar et al. [5] propose a concrete methodology, namely the Goal-Risk framework to analyze and model security problems. It captures the stakeholders' goals, risks that might threaten the goals, and countermeasures required to mitigate the unacceptable the risk. Similarly, CORAS [14] provides a comprehensive method for managing risk (i.e., not only information security risk). The CORAS analysis is centered on analyzing "unwanted incidents" for a defined asset model. When the risk level of those unwanted incidents is beyond the acceptable one, several treatments will be introduced to the system. Though these frameworks support the risk assessment process of possible threats, the elicitation of risk (i.e., event in the GR or unwanted incident in the CORAS) is a manual process which depends on the analysts' knowledge and there is no technique to ensure that the identification of threats is complete and exhaustive. Through our approach, analysts can identify automatically the threats that exist in a given organization and permission setting. However, the outcome (i.e., threats) of our analysis can be considered as an input for further risk assessment.

VII. Conclusion and Future work

In this paper we propose a framework to automatically identify threats that is complementary to other threats identification approaches that rely on the analyst level of expertise such as risk assessment. The threats are derived from potential misuse of actors' permissions on resources. In order to identity potential threats, we have extended SI* with: 1) relationships between resources, 2) actors' permissions on such resources, 3) security requirements on goals and resources, and 4) delegation of a particular permission type to the other actor.

We are planning to extend the framework, by considering other relationships between resources and identify more complex threat patterns that lead to the violation of security properties.

Acknowledgments

This work has been partially funded by NFSC of China (Grant No.60873064), the 973 Program (Grant No.2009CB320706), EU-FP7-ICT-IP-ANIKETOS (Grant No.257930), EU-FP7-ICT- IP-SecureChange (Grant No.231101), and EU-FP7-ICT-NoE-NESSoS project (Grant No.256980).

References

[1] F. Massacci, J. Mylopoulos, and N. Zannone, "Security Requirements Engineering : The SI * Modeling Language and the Secure Tropos Methodology," in *Advances in Intelligent Information Systems*, ser. Studies in Computational Intelligence, Z. Ras and L.-S. Tsay, Eds. Springer Berlin / Heidelberg, 2010, vol. 265, pp. 147–174.

[2] A. Van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models," *Proceedings. 26th International Conference on Software Engineering*, pp. 148–157, 2004.

[3] L. Liu, E. Yu, and J. Mylopoulos, "Security and privacy requirements analysis within a social setting," *Proc.of RE*, vol. 3, pp. 151–161, 2003.

[4] G. Elahi, E. Yu, and N. Zannone, "A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities," *Requirements Engineering*, vol. 15, no. 1, pp. 41–62, Nov. 2009.

[5] Y. Asnar, P. Giorgini, and J. Mylopoulos, "Goal-driven risk assessment in requirements engineering," *Requirements Engineering*, vol. 16, no. 2, pp. 101–116, 2011.

[6] E. Yu, "Modelling strategic relationships for process reengineering," Ph.D. dissertation, University of Toronto, Canada, 1995.

[7] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Modeling security requirements through ownership, permission and delegation," in *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, no. July. IEEE, 2005, pp. 167–176.

[8] Y. Asnar, R. Bonato, P. Giorgini, F. Massacci, V. Meduri, C. Riccucci, and A. Saidane, "Secure and Dependable Patterns in Organizations: An Empirical Approach," in *Requirements Engineering, 2007. Proceedings. 15th IEEE International Conference on*, 2007.

[9] T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer, "Aggregate functions in disjunctive logic programming: Semantics,complexity,and implementation in dlv," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI) 2003*. Elsevier Science Publishers, 2003, pp. 847–852.

[10] K. Yskout, R. Scandariato, B. De Win, and W. Joosen, "Transforming security requirements into architecture," in *Third International Conference on Availability, Reliability and Security, 2008. ARES 08*. IEEE, Mar. 2008, pp. 1421–1428.

[11] A. Van Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering," *IEEE Transactions on Software Engineering*, vol. 26, no. 10, pp. 978–1005, 2000.

[12] T. Li, L. Liu, and B. R. Bryant, "Service Security Analysis Based on i*: An Approach from the Attacker Viewpoint," in *Security, Trust, and Privacy for Software Applications (STPSA 2010)*, Seoul, 2010, pp. 127–133.

[13] N. Mayer, P. Heymans, and R. Matulevicius, "Design of a modelling language for information system security risk management," in *Proceedings of the 1st International Conference on Research Challenges in Information Science (RCIS 2007)*, 2007, p. 121–131.

[14] M. S. Lund, B. Solhaug, and K. Stolen, *Model-Driven Risk Analysis - The CORAS Approach*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.