Detecting Insider Threats: a Trust-Aware Framework

Francisco Moyano, Carmen Fernandez-Gago
Network, Information and Computer Security Lab
University of Malaga, 29071 Malaga, Spain
{moyano,mcgago}@lcc.uma.es

Federica Paci
University of Trento, Italy
paci@disi.unitn.it

September 20, 2013

Abstract

The number of insider threats hitting organizations and big enterprises is rapidly growing. Insider threats occur when trusted employees misuse their permissions on organizational assets. Since insider threats know the organization and its processes, very often they end up undetected. Therefore, there is a pressing need for organizations to adopt preventive mechanisms to defend against insider threats.

In this paper, we propose a framework for insiders identification during the early requirement analysis of organizational settings and of its IT systems. The framework supports security engineers in the detection of insider threats and in the prioritization of them based on the risk they represent to the organization. To enable the automatic detection of insider threats, we extend the SI* requirement modeling language with an asset model and a trust model. The asset model allows associating *security properties* and *sensitivity* levels to assets. The trust model allows specifying the *trust level* that a user places in another user with respect to a given permission on an asset. The insider threats identification leverages the trust levels associated with the permissions assigned to users, as well as the sensitivity of the assets to which access is granted. We illustrate the approach based on a patient monitoring scenario.

1 Introduction

As reported by the 2011 CyberSecurity Watch Survey, 21% of cyber crimes were committed by insiders [6]. However, the 46% of the respondents thought that damage caused by insider attacks was more severe than damage from outsider attacks. In fact, insider attacks can cause significant damage to the affected organizations e.g loss of money, loss of reputation, or loss of customers, among others. As defined by CERT [17], an insider is “a current or former employee, contractor, or business partner who has or had authorized access to an organization’s network, system, or data and intentionally exceeded or misused that access in a manner that negatively affected the confidentiality, integrity, or availability of the organization’s information or information systems”. Insider attacks are more difficult to detect because they are trusted employees who have legitimate and often privileged access to critical or valuable assets, and have knowledge of the organization and of its processes. Thus, to defend from insider threats, preventive measures need to be taken that detect and assess the risks associate with insiders rather than reactive measures after the attack has been conducted.

In this paper, we present an approach to assist security engineers in the detection of insider threats during the early security requirements analysis phase of a socio-technical system development life cycle. Our approch is complementary to other threats identification approaches that rely on the analyst level of expertise such as risk assessment. With our approach, the security engineer can identify automatically the insider threats that exist in a given organization and permission setting and assess the associated risks. The approach consists in first modeling in the SI* requirements modeling language [12] the system stakeholders, their goals, their assets, the security properties (e.g confidentiality, integrity, availability) that stakeholders want to hold for their assets, the permissions that the stakeholders have on assets, and delegation and trust of permissions relationships among them. Trust of permission relationships represent the belief of the grantor of a permission on an asset that the grantee will not misuse it: an agent can be either trusted with a permission or distrusted. The level of *trust* associated with an agent with respect to a granted permission is crucial to assess the risk of the agent being an insider threat: the lower the level of trust associated with a permission is, the higher is the likelihood that the agent will misuse the permission.

To support the automatic detection of insider threats, we extend the SI* requirements modelling language proposed in [2] with an *asset* and *trust* model. The asset model associates with assets a *sensitivity* value that represent how valuable the asset is for the owner. The trust model associates different levels of trust (e.g. high, medium and low trust) with a permission granted to an agent rather than a single binary value (trusted, not trusted) as currently supported by the SI* language. Based on the sensitivity and trust levels, we define a set of rules to automatically identify insider threats to an asset and prioritize them based on the risk associated with the threat. The risk associated with the insider threat is given by the likelihood that the threat occurs that is quantified by the *trust level* associated with the permission granted to the insider agent, and the cost of the permission being misused that is quantified by the *sensitivity* of the asset being harmed.

The rest of the paper is organized as follows. Section 2 introduces a running example taken from the healthcare domain. We introduce the SI* framework and its extensions proposed in [2] in Section 3. We present the asset and the trust model in Section 4 and the process to identify and prioritize insider threats in Section 5. We discuss the related work in Section 6 and outline future work in Section 7.

2 Running Example - Patient Monitoring

To illustrate our framework, we use a patient monitoring scenario from the eHealth case study proposed in the NESSoS European project [1]. The scenario involves five 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. The **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. The **Pharmacy** is responsible for managing drugs and provide them to the patients. All the information about drugs is stored in the pharmacy's computer. The **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. Finally, the **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.

3 The SI* Modeling Framework

The SI* modeling language [12] has been proposed to capture security and functional requirements of socio-technical systems. SI* is founded on the concepts of *agent*, *role*, *service*, and relations such as *AND/OR decomposition* and *means-end*. An *agent* is an active entity with concrete manifestations and is used to model humans as well as software agents and organizations. A *role* is the abstract characterization of the behavior of an active entity within some context. The term *service* is used to denote a *goal*, a *task* and a *resource*. A *goal* captures a strategic interest that is intended to be fulfilled. A *task* represents a particular course of actions that produces a desired effect. It can be executed to satisfy a goal. A *resource* is an artifact produced/consumed by a goal or a task. *AND/OR decomposition* is used to refine a goal, while *means-end* identifies goals that provide means for achieving another goal or resources produced or consumed by a goal/task.

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. *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 order to support the identification of threats at organizational level, in [2] SI* has been extended to represent different types of actors' permissions on resources and different types of relationships between resources. Goals and resources are considered

[1] http://www.nessos-project.eu/

Table 1: 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

as assets that need to be protected because they bring value 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. The permission type granted on a resource determines the type of actions an actor can perform on a resource (see Table 1). Thus, a permission type might yield to the violation of a specific security property if the actor misuses the actions granted by a permission type. Moreover, a given permission granted on a resource can be extended to other resources that are related to the resource by the relations reported in Table 2.

Table 2: Relationships between resources

Relationship	Description
$store\_in$	captures a situation where an informational resource is stored in a physical resource
$part\_of$	indicates that a resource consists of other resources.
$require$	denotes that a resource might require another resource to function.

In order to allow the analysis of SI* models, the semantics of SI* has been defined in the Answer Set Programming (ASP for short) paradigm which is a variant of Datalog with negation as failure and disjunction. This paradigm supports specifications expressed in terms of facts and Horn clauses, which are evaluated using the stable model semantics. Here, SI* models are encoded as sets of facts. Rules (or axioms) are Horn clauses that define the semantics of SI* concepts. To support the formalization in ASP, the DLV inference engine is used. Table 3 summarizes the predicates to formalize an SI* model in ASP.

Example 1. _Figure 1 shows the SI* model for the Patient Monitoring scenario. The model consists of five roles: the Hospital, the Patient, the Pharmacy, the Pharmacist, and the Drug Manager. Patient (_Role_) can be played by three agents Bob, Kate, and Jane. The Patient (_**O**wns_) the resources Patient data and Prescription. It delegates to the Hospital the manage permission on Patient data, and it delegates the access permission on Prescription to the Pharmacy. The Pharmacy has the intention (_**R**equest_) to fulfill the goal Sell drug which is (_AND-decomposed_) into subgoals Manage drug and Provide drug: the fulfillment of Manage drug is delegated to the_

Table 3: Predicates for ASP SI* formalization

Goal model

service(Service:s)
goal(Goal:g)
resource(Resource:r)
actor(Actor:x)
agent(Agent:a)
role(Role:p)
play(Agent:a,Role:p)
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)

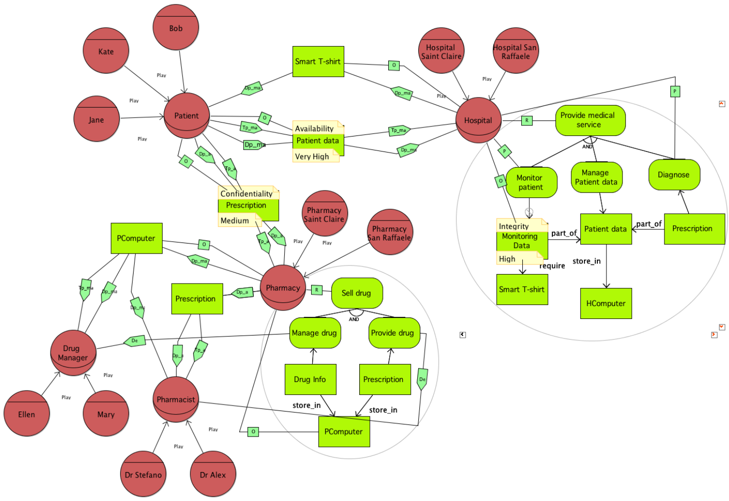


Figure 1: Example of SI* model - Patient Monitoring

Legend: The circles denote roles or agents, the ovals denote goals, while the rectangles represent resources. **Dp_a**, and **Dp_ma** represent delegation of permission relation where the permission type is access and manage respectively. Similarly,**Tp_a**, and **Tp_ma** represent trust of permission relation where the permission type is access and manage. Services that are considered assets are labeled with the security property that should be satisfied and their sensitivity level.

Drug Manager *while the fulfillment of* Provide drug *is delegated to the* Pharmacist. *The* Pharmacy (**O**wns) *the resource* PComputer. *It grants to* Drug Manager *the* *manage permission on PComputer and the access permission on Prescription to the Pharmacist. The Hospital (*Role*) has an intention (***R**equest*) to fulfill the goal Provide medical service which is (*AND-decomposed*) into subgoals Monitor patient, Manage patient data, and Diagnose. Some goals can produce or consume resources. For example, the goal Diagnose requires the resource Patient data and produces the resource Prescription. The Hospital (***O**wns*) the resource Smart T-shirt and delegates to the Patient the manage permission on it.*

4 SI* extensions

In this section we present the two main extensions that we propose to SI*: *asset model* and *trust model*.

4.1 Asset Model

We consider an asset a service for which the owner specifies the sensitivity and a security property that expresses the need of protecting the service. We introduce a predicate *sec_req(s,sp,p)* to specify the security property *sp* that should be preserved for a service *s* owned by a role *p*. Similarly, to denote that that a security property *sp* should be preserved for a specific instance of a service *s*, *service_instance(s, a, p)*, we introduce the predicate *sec_req(service_instance(s, a, p),sp,a,p)*. We also introduce the predicate *sensitivity(s,sl,p)* to indicate that a service *s* owned by role *p* has sensitivity level *sl*, and the predicate *sensitivity_instance(service_instance(s, a, p),sl,a,p)* to associate a sensitivity level *sl* to an instance of the service *s* owned by the agent *a* playing the role *p*. To denote at organizational level that a service is an asset, we introduce the predicate *asset(s, p)* where *s* is a service and *p* is the role who owns it. Instead, the predicate *asset_instance(service_instance(s, a, p),a,p)* holds when an instance of a service *s* is an asset.

Table 4: Formalization of SI* extensions

Asset model
sec_req(Service:s, SProperty:sp,Role:p)
sec_req_instance(ServiceInstance:si, SProperty:sp,Agent:a,Role:p)
asset(Service:s, Role:p)
asset_instance(ServiceInstance:si, Agent:a,Role:p)
sensitivity(Service:s, SLevel:sl, Role:p)
sensitivity_instance(ServiceInstance:si, SLevel:sl, Agent:a,Role:p)
asset(S,P) ← sec_req(S,SP,P) ∧ sensitivity(S, SL, P)
Trust model
trust_perm(Role:p, Role:p1, Service:s, PType:pt)
trust_perm_instance(Agent:a, Agent:a1, ServiceInstance:si, PType:pt, TLevel:tl)

In our model we distinguish between two types of assets: *direct* and *indirect* assets. *Direct assets* are services for which a security property and a sensitivity level are explicitly modeled in the SI* model, while *indirect assets* are services for which the security property and the sensitivity level is determined based on the relations with other services. The identification of indirect assets is based on a set of rules (reported in Table 5) that consider the relations among resources - *stored_in*, *part_of* and *require*

Table 5: Axioms for identifying indirect assets

Indirect Assets Identification	
S1	$sec\_req(R, SP, P) \leftarrow store\_in(R1, R) \wedge sec\_req(R1, SP, P)$
S2	$sec\_req(R, integrity, P) \leftarrow store\_in(R1, R) \wedge sec\_req(R1, confidentiality, P)$
S3	$sec\_req(R1, SP, P) \leftarrow part\_of(R1, R) \wedge sec\_req(R, SP, P)$
S4	$sec\_req(R, integrity, P) \leftarrow require(R1, R) \wedge sec\_req(R1, integrity, P)$
S5	$sec\_req(R, availability, P) \leftarrow require(R1, R) \wedge sec\_req(R1, availability, P)$
S7	$sec\_req(G, SP, R, P) \leftarrow secure\_req(R, SP, P) \wedge means\_end(G, R)$
S8	$sec\_req(G, confidentiality, R, P) \leftarrow secure\_req(R, confidentiality, P) \wedge means\_end(R, G)$
S9	$sec\_req(G1, SP, R, P) \leftarrow subgoal(G1, G) \wedge sec\_req(G, SP, R, P)$
S10	$sensitivity(R, SL, P) \leftarrow store\_in(R1, R) \wedge sensitivity(R1, SL, P)$
S11	$sensitivity(R, SL, P) \leftarrow part\_of(R1, R) \wedge sensitivity(R1, SL, P)$
S12	$sensitivity(R, SL, P) \leftarrow require(R1, R) \wedge sensitivity(R1, SL, P)$
S13	$sensitivity(G, SL, P) \leftarrow means\_end(G, R1) \wedge sensitivity(R1, SL, P)$

- and the relationship *means_end* among the resources and the goals that requires the resources to be fulfilled. We assume that if an asset is related to a service by one of these relations, the same security property should hold for the asset and service (axioms $S1$-$S9$) and that the two assets should have the same sensitivity level (axioms $S10$-$S13$). If the direct asset is related to another direct asset only the security property is propagated to the other asset.

Example 2. *In Figure 1 Prescription is an medium sensitive asset for the Patient, who requires that Confidentiality of Prescription is preserved. Since Prescription is stored_in PComputer, also the Confidentiality of PComputer needs to be preserved.PComputer is an indirect asset which has the same sensitivity of Prescription.*

4.2 Trust Model

The SI* trust model is very simple since it supports only binary trust values: either an agent is trusted or is distrusted for a given permission on a resource. However, in real scenarios trust is not a binary value but an agent can be assigned different levels of trust. The different values of trust depend very much on the used definition, being the latter very much dependent on the context. There is no a standard definition of trust. However, usually, a trust relationship holds between two agents: a trustor (the one who places trust) and a trustee (the one performing a given action and to who(m) the trustor places trust in). In this paper, we define trust as the expectation that the trustor places on the trustee on a specific context for performing a specific task. The context in our case is a permission that is granted to the trustee on a given asset. The trust level is important to determine the likelihood that the trustee will misuse the granted permission to harm the asset.

Our intention is to extend SI* with a trust model that associates a *trust level* with a trust of permission relation between two agents. The trust levels are then translated into trust labels that are used to define insider threats identification rules, which determine if an agent misuses a granted permission on an asset and the risk associated to the threat.

First, we assume that trust levels can be represented in two forms: labels - e.g. *Very Good*, *Good*, *Neutral*, *Bad*, *Very Bad*- and numbers in the interval [0, 1], as in the trust model proposed in [7]. Then, the second important step to define a trust model for SI* is to set a way to derive trust values. The assignment of trust values can be done in a discrete way by using for example 0 for representing non-trust at all; 1 for representing total trust and 0.5 for representing an intermediate value. The way of calculating trust can be made in a more accurate way by using some kind of mathematical or statistical functions. For our purpose, the way trust values are calculated is not very relevant. Jøsang provides a comprehensive review on trust models [8]. We assume that some trust values are already assigned to trust of permission relationships between agents in the SI* model and that these values are leveraged by the organization stakeholders in order to compute trust values for pairs of agents for which such relationships are not explicitly modelled. Thus, to determine the trust level that an agent A places on another agent B regarding how B will behave with respect to a granted permission, we leverage the trust of permission relationships that other agents have with A and B. Incorporating a trust model in SI* thus requires not only to extend trust of permission relationships with trust values and permissions but also to add the following rules:

- *Derivation rules* - generate a trust value for agents A and B given the trust values that other agents trusted by A have in B
- *Resolution rules* - resolve potential conflicts that may arise from applying the derivation rules
- *Transformation rules* - specify how to translate trust values to trust labels.

We formally introduce these rules as follows.

4.2.1 Derivation and Resolution Rules

These rules aim to exploit trust levels associated with trust of permission relationships present in an SI* model in order to compute trust values for trust of permission relationships not explicitly represented in the model. These rules are supported by the concept of trust evaluation. This concept and those of trust statement, and linear and consensus functions are introduced in [1].

Definition 1 (Trust Evaluation)**.** *A trust evaluation is a function $\mathcal{F} : E \times E \times C \longrightarrow TD$, where E is a set of agents, C is a set representing the context in which the trust evaluation can take place, and TD is the trust domain.*

Note that the definition is generic and applicable to different settings and for different purposes. In this paper, the output of this function is the trust value in the extended *trust_perm* relationship that measures quantitatively the expectation of an agent about the behaviour of another agent with respect to a given permission. In this work, we instantiate the context C as the permission granted to the trustee on a given asset. Before we introduce these extensions, *trust_perm* relation did not reflect this quantitative measurement, since it was an atomic relationship: either it was present in the SI* model or not.

Once the requirement engineer has designed the SI* model of the socio-technical system and its instances, there might appear two types of relationships configurations. Depending on the configuration that we have there might be two different types of trust evaluation functions. We may use a linear trust function, or a consensus trust function. Yet before providing their formal definitions, we need to define what a *trust statement* is.

Definition 2 (Trust Statement)**.** *A trust statement is an element* $(Trustor, Trustee, Context, Value) \in E \times E \times C \times TD$*, where* E *is the set of all entities in the system;* C *is a set representing a context; and* TD *is a Trust Domain.*

Trust of permission relationships are a particular instance of trust statements where $Context$ is the permission granted to the trustee on an asset instance and $Value$ is the trust level placed by the trustor in the trustee for the permission. To represent trust statements we have introduced the predicate *trust_perm_instance(A, A1,asset_ instance (service_instance (S,A,P)), PT, TL)*. Trust statements can form trust chains. Linear and consensus trust functions evaluate trust over these chains, as defined next.

Definition 3 (Linear Trust Function)**.** *A linear trust function is a function,* $f : \bigcup_{n=2}^{\infty} \overbrace{TD \times \cdots \times TD}^{n} \longrightarrow TD$*, that calculates the trust level associated to a path or chain of trust statements, such that* $f(v_1, \ldots, v_n) = 0$ *if, and only if,* $v_i = 0$ *for any* $i \in \{1, \ldots, n\}$*, where* $v_i \in TD$ *and* TD *is a trust domain.*

Definition 4 (Consensus Trust Function)**.** *A consensus trust function is used to calculate the trust level associated to a set of paths or chains of trust statements. It is defined as,* $g : \bigcup_{n=2}^{\infty} \overbrace{TD \times \cdots \times TD}^{n} \longrightarrow TD$*, where* TD *is a trust domain and*

1. $g(z_1, \ldots, z_{i-1}, z_i, z_{i+1}, \ldots, z_n) = g(z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n)$ *if* $z_i = 0$
2. $g(z) = z$

As a consequence of applying these functions to an SI* model, an agent might end up holding several trust of permission relations with a given agent. However, it would be optimal if an agent only holds one value for any other agent of the system. Resolution functions could solve this.

Definition 5 (Trust Resolution Function)**.** *A trust resolution function is a function,* $f : \bigcup_{n=2}^{\infty} \overbrace{TD \times \cdots \times TD}^{n} \longrightarrow TD$*, such that* $f(v_1, \ldots, v_n) \leq max(v_1, \ldots, v_n)$ *and* $f(v_1, \ldots, v_n) \geq min(v_1, \ldots, v_n)$*, where* $v_i \in TD$ *and* TD *is a trust domain.*

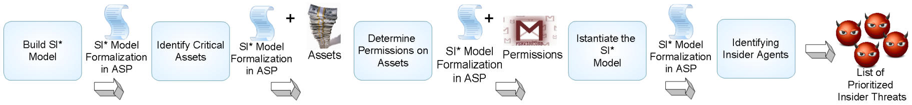

Build SI* Model
SI* Model Formalization in ASP
Identify Critical Assets
SI* Model Formalization in ASP
+
Assets
Determine Permissions on Assets
SI* Model Formalization in ASP
+
Permissions
Istantiate the SI* Model
SI* Model Formalization in ASP
Identifying Insider Agents
List of Prioritized Insider Threats

Figure 2: Process to Identify Insider Threats

Basically, given a set of trust values, the resolution function produces one unique representative trust value that is upper bounded by the maximum and lower bounded by the minimum of the original trust values. Derivation and resolution rules rely on functions to compute the trust values. For this purpose, different functions could be used (e.g. maximum, minimum, arithmetic or geometric means).

4.2.2 Transformation Rules

Once we obtain the final numeric values for every trust of permission relationship, transformation rules must be used in order to translate these values, which are in an interval $[a, b]$ ([0, 1] is the chosen one in this case) into a label in a given set of labels that forms a trust scale [7].

Definition 6 (Trust Scale)**.** *A trust scale for a given context $c \in C$ is composed of an ordered set $\mathcal{L}^c$ of trust labels L^c_i, where $i \in I_c$, that represent discrete trust meanings; and a trust evaluation that is an increasing function, $f : \mathcal{L}^c \to (0, 1]$, such that $f(L^c_{n_c}) = 1$.We denote $x^c_i := f(L^c_i)$ and $x^c_0 := 0$.*

Example 3. *Let us consider the SI* model illustrated in Figure 1 that shows the patient monitoring scenario. Let us suppose that the agent Bob (playing the Patient role) wants to determine the level of trust with which he can grant the access permission on its asset Prescription to Ellen (playing the Drug Manager role). Since Bob has no direct trust relationship with Ellen we need to evaluate the trust value that Bob places in Ellen based on the following trust chain:* trust_ perm_ instance (Bob, Pharmacy Saint Claire, asset_ instance (service_ instance (Prescription, Bob, Patient), Bob, Patient), access, very good) *;* trust_perm_instance(Pharmacy Saint Claire, Ellen,asset_instance(service_instance(Prescription,Bob,Patient), Bob, Patient), access, good). *Note that Ellen has access to the instance of Prescription owned by Bob because it is stored in the instance of PComputer owned by Pharmacy Saint Claire on which Ellen has been granted manage permission with good trust level and having the manage implies the access permission. Let us assume that the trust scale and the trust evaluation function are defined as follows:*

- *Very good → 1*
- *Good → 0.8*
- *Neutral → 0.6*
- *Bad → 0.4*

- *Very Bad → 0.2*

*Let us also assume that in order to compute the trust value that Bob can place in Ellen for the access permission we use the product as consensus function. Thus, the trust level for Ellen is 1 * 0.8 = 0.8 which corresponds to label Good. Thus, we can add to the SI* model formalization the following trust of permission relationship between Bob and Ellen:* trust_perm_instance (Bob, Ellen, asset_ instance(service_instance(Prescription, Bob, Patient), Bob, Patient), access, good).

5 Identifying Potential Threats

In this section we present the steps to follow for the process for identifying insider threats (see Figure 2).

5.1 Build SI* Model

This step aims to create an SI* model, which captures all the stakeholders of the system modeled as agents, the role that they play inside the organization, their goals, tasks and resources, the relations between resources, and the delegation and trust relationships among the roles. To enable the automatic detection of insider threats, we need to translate the graphical SI* model into formal specifications in ASP (see Section3).

Example 4. *The SI* model in Figure 1 is produced during this step. A snapshot of the formalization of the model in ASP is reported in Table 6.*

5.2 Identify Critical Assets

This step aims to identify those services in the SI* model that are assets. First, direct assets are identified by labelling the critical services with the security property that the role owning the service wants to hold for it, and with the service's sensitivity level (see Section 4). Then, the indirect assets are determined based on the set of rules in Table 5.

Example 5. *In Figure 1 there are three direct assets owned by the Patient role: Prescription, Patient Data, and Monitoring Data. The Patient requires* confidentiality *to hold for Prescription,* availability *should hold for Patient Data, while* integrity *should be satisfied for Monitoring Data. Smart T-Shirt, HComputer, PComputer, Diagnose, Manage Patient data, Monitor Patient, and Provide Drug are indirect assets. For example, PComputer is an indirect asset because the asset Prescription is stored in PComputer and thus the* confidentiality *of PComputer needs to be preserved. Similarly, the goal Diagnose is an indirect asset because it is linked to the asset Prescription by a means_ end relation, and thus also the confidentiality of the goal needs to hold.*

Table 6: ASP formalization for SI* model

```
role(patient)
role(hospital)
goal(provide_medical_service)
goal(monitor_patient)
goal(diagnose)
..........
resource(monitoring_data)
resource(patient_data)
resource(prescription)
resource(computer)
resource(smart_t_shirt)
..........
means_end(diagnose,prescription)
resource(smart_t_shirt)
del_perm_manage(smart_t_shirt,patient)
del_perm_manage(hospital,smart_t_shirt)
own(hospital,smart_t_shirt)
del_perm_manage(patient_data,hospital)
own(patient,patient_data)
role(pharmacy)
..........
own(pharmacy,pcomputer)
del_perm_manage(pcomputer,drug_manager)
..........
del_perm_access(prescription,pharmacy)
own(patient,prescription)
del_perm_access(pharmacy,prescription)
..........
agent(kate)
agent(bob)
play(kate,patient)
play(bob,patient)
```

Table 7: ASP rules to instantiate the SI* model

Istantiating Assets	
$A1$	$sec\_req\_instance(service\_instance(S, A, P), SP, A, P) \leftarrow sec\_req(S, SP, P) \wedge service\_instance(S, A, P) \wedge instance(A, P)$
$A2$	$sensitivity\_instance(service\_instance(S, A, P), SL, A, P) \leftarrow service\_instance(S, A, P) \wedge instance(A, P) \wedge sensitivity(S, SL, P)$
$A3$	$asset\_instance(service\_instance(S, A, P), A, P) \leftarrow sec\_req\_instance(service\_instance(S, A, P), SP, A, P) \wedge sensitivity\_instance(service\_instance(S, A, P), SL, A, P)$
Istantiating Permissions	
$A4$	$permission\_instance(A, service\_instance(S, A, P) \leftarrow permission(P, S, PT) \wedge instance(A, P) \wedge service\_instance(S, A, P)$

5.3 Determine Permissions on Assets

This step determines the permissions that roles are granted on assets. The permissions are assigned to roles based on a set of axioms that take into account if a role is the owner of a resource and of the relations between resources - *stored_in*, *part_of* and *require*. The axioms assume the owner of a resource has the highest permission on a resource (i.e., *manage*) or that a role with the *manage* permission on a resource can delegate any permission type on the resource to another actor. In addition, if a role has a *manage* permission on an resource which stores another resource, s/he then has the *manage* permission also on the stored resource. Last, if a role has a permission on a resource, then s/he has the same permission on each subpart of the resource. For a complete list of the axioms, we refer the reader to [2].

Example 6. *The Patient has delegated the access permission to the Pharmacy on Prescription, and thus the Pharmacy has the access permission on Prescription. Moreover, the Pharmacy has the manage permission on PComputer and thus it has also the manage permission on Drug Info and Prescription that are stored in PComputer. The Pharmacist and the Drug Manager are granted by Pharmacy the* manage *permission on the PComputer. In addition, the Pharmacist also gains access permission on the Prescription from the Pharmacy. Since the Drug Manager has manage permission on the PComputer and Prescription is stored in PComputer, Drug Manager has manage permission on Prescription.*

5.4 Instantiating the SI* model

This step instantiates the SI* organizational model. We only report the axioms to instantiate the elements of the SI* model that are relevant for the insider threat identification. A complete list of the ASP rule to instantiate an SI* model can be found in [21]. In the following, we introduce the rules to instantiate assets, agents' permissions on assets and the trust of permission relations between agents.

5.4.1 Instantiate Assets

Each instance of an asset is identified with its sensitivity level. The identification is based on the rules given in Table 7. $A1$ states that if a security property holds for a service at organizational level, this property should hold for each instance of that service. $A2$ associates a sensitivity level to an asset instance: the asset instance has the same sensitivity of the asset at organizational level. $A3$ determines if a service instance is an asset: a service instance is an asset if there is a security property that holds for the service instance and the service instance has sensitivity level.

Example 7. *Prescription is an asset owned by the role Patient. The Patient role is played by the agents Bob, Kate, Jane, thus each of them owns one of the following instances of Prescription:*

- asset_instance(service_instance(Prescription,Bob,Patient), Bob,Patient),
- asset_instance(service_instance(Prescription,Kate,Patient), Kate, Patient),
- asset_instance(service_instance(Prescription,Jane,Patient), Jane, Patient).

5.4.2 Instantiate Permissions on Assets

This steps identifies the permissions that agents have on assets. $A4$ states that an agent playing a role inherits the permissions that the role is granted on assets.

Example 8. *The Pharmacy role delegates the manage permission on PComputer to role Drug Manager. Since the Pharmacy is played by the agent Pharmacy San Raffaele, and the Drug Manager is played by agents Ellen and Mary, Ellen and Mary are granted the manage permission on the instance of PComputer owned by Pharmacy San Raffaele.*

5.4.3 Instantiate Trust of Permissions relation

In this step the trust of permission relationship between agents owning assets and agents having permissions on their assets are identified. This implies to determine the level of trust that the owner places in the other agent for the granted permission: the trust value can be already given or it can be computed based on a trust chain as described in Section 4.

Example 9. *At organizational level the* Pharmacy *trusts the role* Drug Manager *with the* manage *permission on* Prescription*. This relationship needs to be instantiated for each instance of* Prescription *asset and each agent playing the* Pharmacy *and* Drug Manager *roles. Let suppose that we want to instantiate the trust of permission relationships for the* Prescription *instance owned by* Bob *- asset_instance(service_instance(Prescription, Bob, Patient), Bob, Patient).* Bob *trusts* good *both* Pharmacy Saint Claire *and* Pharmacy San Raffaele *with the* access *permission on asset_instance (service_ instance(Prescription, Bob, Patient), Bob, Patient). On their turn,* Pharmacy San Raffaele *and* Pharmacy Saint Claire *places the following trust levels in the agents* Dr Alex *and* Dr Stefano *for the* access *permission on asset_instance(service_instance (Prescription, Bob, Patient, Bob, Patient)):*

- trust_perm_instance(Pharmacy Saint Claire, Dr Stefano,asset_instance (service_instance (Prescription, Bob, Patient), Bob, Patient), access, neutral)
- trust_perm_instance(Pharmacy San Raffarele, Dr Alex, asset_instance(service_instance(Prescription,Bob, Patient), Bob, Patient), access, good)

The trust of permission relations between Bob *and* Dr Alex *and* Bob *and* Dr Stefano *can be computed as described in Example 3. As result of the computation the following trust of permission relationships can be added to the ASP formalization of the SI* model:*

- trust_perm_instance(Bob, Dr Stefano, asset_instance(service_instance(Prescription, Bob, Patient), Bob, Patient), access, neutral) *and*
- trust_perm_instance(Bob, Dr Alex, asset_instance(service_instance(Prescription,Bob, Patient), Bob, Patient), access, good).

5.5 Detecting Insider Threats

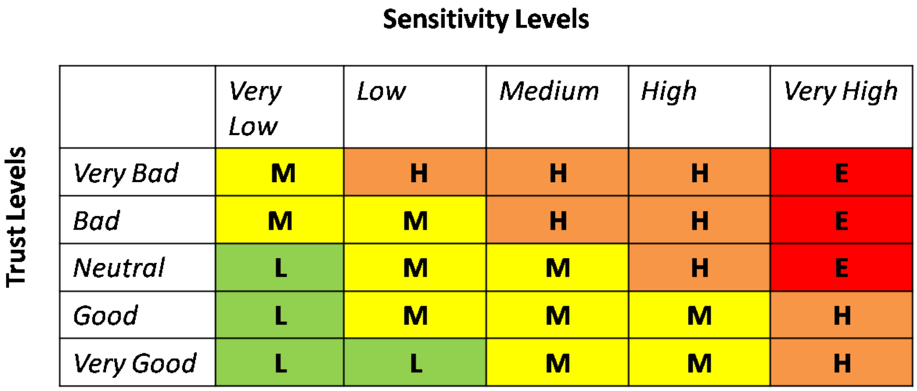
We assume that an agent A is an insider for a given asset S when two conditions hold:

a) A is granted a permission PT on the asset S that is sufficient to violate the security property associated with S;

b) The agent who owns the resource S does not fully trust A with permission PT.

We introduce a *threat* predicate to specify when an agent is an insider for a given instance of an asset and the risk associated with the insider threat.

Typically, the risk associated with a threat is given by the probability that a threat occurs and the severity of the threat. Here, we determine the level of risk associated



Sensitivity Levels

Trust Levels	Very Low	Low	Medium	High	Very High
Very Bad	M	H	H	H	E
Bad	M	M	H	H	E
Neutral	L	M	M	H	E
Good	L	M	M	M	H
Very Good	L	L	M	M	H

Figure 3: Risk Levels

with the threat initiated by an agent A based on two dimensions: *the sensitivity* of the asset S and the *trust level* with which A is granted the permission PT. The sensitivity quantifies the cost of the threat by A, while the trust level quantifies the likelihood that the threat occurs. Intuitively, higher is the sensitivity of the asset S, higher is the damage for the organization. Similarly, higher is the trust level, lower is the likelihood that the agent will misuse the granted permission [4].

Figure 3 is an example of how the risk level of a threat can be determined based on sensitivity and trust levels. The rows of the table represent the trust levels, while the columns represent the sensitivity levels. Each entry of the matrix specifies the risk level for a given combination of sensitivity and trust levels. The risk level can assume one of the following values: Low, Moderate, High, Extreme. How trust and sensitivity relates to each other depends on the organization's policy and should not be fixed beforehand.

The identification of the insider threats and their risk level is based on a set of axioms reported in Table 8. Due to the lack of space, we list only the axioms to detect insider threats to assets' confidentiality, integrity and availability with extreme risk level. Axioms $T1.a$ - $T1.c$ identify insider threats to confidentiality: the insider has *access* permission on the asset being harmed and the owner of the asset places *very bad*, *bad* or *neutral* trust level in the insider for the granted permission. Axioms $T2.a$ - $T2.c$ allow to detect insider threats to integrity: in this case the insider needs to be granted a *modify* permission on the asset and the owner places *very bad*, *bad* or *neutral* trust level for the granted permission. Axioms $T3.a$ - $T3.c$ identifies insider threats to availability.

The modeling and the reasoning based on the above axioms are supported by the SI* tool which is an Eclipse plug-in equipped with a DLV engine. The tool interface allows to draw an SI* model which is automatically translated into ASP specification. The tool also allows to input the rules for insider threat identification so that the problem of identifying insider threats is the same as checking a DLV program that formalize the SI* model and the axioms.

Example 10. *Let us assume we want to determine all the possible insiders for the* instance of Prescription asset owned by the Patient Bob. The reasoning reports the following insiders:

- threat(Dr Stefano,asset_instance(service_instance(Prescription, Bob, Patient),Bob,Patient), confidentiality, moderate)
- threat(Dr Alex,asset_instance(service_instance(Prescription, Bob, Patient),Bob,Patient), confidentiality, moderate)
- threat(Ellen,asset_instance(service_instance(Prescription, Bob, Patient),Bob,Patient), confidentiality, moderate)
- threat(Mary,asset_instance(service_instance(Prescription, Bob, Patient),Bob,Patient), confidentiality, high)
- threat(Ellen,asset_instance(service_instance(Prescription, Bob, Patient),Bob,Patient), availability, moderate) threat(Mary,asset_instance(service_instance(Prescription, Bob, Patient),Bob,Patient), availability, high)

Dr Stefano and Dr Alex are two insiders which represent a moderate risk to the confidentiality of Prescription instance owned by Bob because they have been granted access permission on the asset instance and they are trusted good for such permission by Bob. Ellen and Mary are insiders to both the confidentiality and the availability of Prescription asset owned by Bob because the following conditions hold:

- the asset instance is stored in the instance of PComputer owned by the Pharmacy Saint Claire and Pharmacy San Raffaele
- Pharmacy Saint Claire trusts good Ellen with the manage permission on the instance of PComputer owned by the Pharmacy San Raffaele
- Pharmacy San Raffaele trusts bad Mary with the manage permission on the instance of PComputer owned by the Pharmacy San Raffaele
- Ellen and Mary thus have the same permission on the Prescription asset owned by the Patient Bob stored in the instances of PComputer owned by Pharmacy Saint Claire and Pharmacy San Raffaele respectively
- having the manage permission on an asset implies to have also the access permission on an asset
- Ellen and Mary are trusted Bob good and bad with the manage permission on the instance of Prescription owned by Bob
- manage permission is sufficient to violate the availability of a given asset while the access permission is sufficient to violate the confidentiality of an asset.

6 Related Work

Several proposals have attempted to include security analysis into the requirement analysis process. Among goal-oriented approaches, van Lamsweerde extends KAOS by introducing the notions of obstacle [19] and anti-goal [18] in order to analyse 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), while KAOS anti-goal captures the intention of an attacker. 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. Liu et al. [11] propose an extension of the i* framework [20] to identify attackers, and analyse vulnerabilities through actor's dependency links. In this framework, all actors are considered as potential attackers. Therefore, their capabilities are analysed and possible damages caused by actors are assessed. In Li et al. [10], the authors proposed a formal framework to support attacker analysis. Similarly, Elahi et al. [5] propose i* extensions to model and analyse the vulnerabilities affecting systems requirements. Matulevičius et al. [13] extend the Secure Tropos [16] language to support modelling of security risks and their countermeasures. For that purpose, the authors analyse the concepts and syntax of Secure Tropos and propose some extensions in order to fill the gap required to align this language with the Information System Security Risk Management (ISSRM) model.

Our approach, unlike the previous approaches, supports an automatic reasoning to identify possible insider threats based on the model formalization which can provide a valuable input (i.e. list of prioritized threats) to these frameworks in order to perform further risk assessment.

Other works focus on integrating risk analysis into the requirement analysis process. SQUARE [14] and SREP [15] are two similar processes that support risk assessment as an explicit step to identify security requirements. Asnar et al. [3] propose a concrete methodology, namely the Goal-Risk framework, to analyse and model security problems. The methodology relies on SI* requirements modeling language to capture stakeholders' goals, risks that might threaten the goals, and countermeasures required to mitigate the unacceptable risks. In [2], Asnar extended SI* with the possibility of specifying permissions and relationships on resources. Based on these extensions, they propose a reasoning to identify threats to resources that occur because roles representing classes of stakeholders of a system misuse their privileges. In our work, we extend this work by adding an asset model and a trust model to identify agents - instances of stakeholders of the system - that may cause harm to organizational assets and to prioritize them based on the risk for the organization.

Li et al. [9] propose a security analysis that verifies that a set of security properties like availability and safety are satisfied while delegating access to resources to partially trusted principals. Similarly to us, they consider that delegating permissions on resources to not fully trusted entities can be a source of threats.

7 Conclusion and Future Work

This paper proposes a framework to support security engineers in identifying insider threats during the security requirements analysis phase of a socio-technical system development life cycle. Our framework provides security engineers with a reasoning that automatically produces a list of possible insiders for organizational assets and the risk they may represent to the organization. The reasoning determines if an agent is an insider for an asset and the risk he/she brings about, based on the sensitivity of the asset, the security property specified for it, the permission assigned to the agent on the asset, and the level of trust the asset owner places in the agent for the granted permission. Once the insider threat is identified, the organization is responsible for taking the necessary countermeasures.

We are aware that our framework has some limitations. First, the validity of the results of the reasoning strictly depends on the quality and completeness of the SI* model, which in turn depends on the level of expertise of the requirements engineer. Second, the visual notation of SI* might not scale well for complex application scenarios.

We are planning to evaluate the strengths and limitations of our framework by conducting a controlled experiment where master students and professionals apply the framework to a real industrial application scenario.

Acknowledgements

This work has been partially funded by the European Commission through the FP7 project NESSoS under grant agreement number 256980. The third author is funded by the Spanish Ministry of Education through the National F.P.U. Program.

References

[1] Isaac Agudo, M. Carmen Fernández Gago, and Javier Lopez. A model for trust metrics analysis. In *TrustBus*, pages 28–37, 2008.

[2] Yudis Asnar, Tong Li, Fabio Massacci, and Federica Paci. Computer aided threat identification. In *Proceedings of the 2011 IEEE 13th Conference on Commerce and Enterprise Computing*, CEC '11, pages 145–152, Washington, DC, USA, 2011. IEEE Computer Society.

[3] Yudistira Asnar, Paolo Giorgini, and John Mylopoulos. Goal-driven risk assessment in requirements engineering. *Requirements Engineering*, 16(2):101–116, 2011.

[4] Jason Crampton and Michael Huth. Towards an access-control framework for countering insider threats. In Christian W. Probst, Jeffrey Hunker, Dieter Gollmann, and Matt Bishop, editors, *Insider Threats in Cyber Security*, volume 49 of *Advances in Information Security*, pages 173–195. Springer US, 2010.

[5] Golnaz Elahi, Eric Yu, and Nicola Zannone. A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities. *Requirements Engineering*, 15(1):41–62, November 2009.

[6] Software Engineering Institute. 2011 cybersecurity watch survey. Technical report, Software Engineering Institute, Carnegie Mellon, 2011.

[7] Javier Lopez Isaac Agudo, Carmen Fernandez-Gago. A scale based trust model for multi-context environments. *Computers and Mathematics with Applications*, 60:209–216, July 2010.

[8] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, March 2007.

[9] Ninghui Li, John C. Mitchell, and William H. Winsborough. Beyond proof-of-compliance: security analysis in trust management. *J. ACM*, 52(3):474–514, May 2005.

[10] Tong Li, Lin Liu, and Barrett R Bryant. Service Security Analysis Based on i*: An Approach from the Attacker Viewpoint. In *Security, Trust, and Privacy for Software Applications (STPSA 2010)*, pages 127–133, Seoul, 2010.

[11] Lin Liu, E Yu, and John Mylopoulos. Security and privacy requirements analysis within a social setting. *Proc.of RE*, 3:151–161, 2003.

[12] Fabio Massacci, John Mylopoulos, and Nicola Zannone. Security Requirements Engineering : The SI * Modeling Language and the Secure Tropos Methodology. In Zbigniew Ras and Li-Shiang Tsay, editors, *Advances in Intelligent Information Systems*, volume 265 of *Studies in Computational Intelligence*, pages 147–174. Springer Berlin / Heidelberg, 2010.

[13] Raimundas Matuleviius, Haralambos Mouratidis, Nicolas Mayer, Eric Dubois, and Patrick Heymans. Syntactic and semantic extensions to secure tropos to support security risk management. *Journal of Universal Computer Science*, 18(6):816–844, mar 2012.

[14] Nancy R. Mead and Ted Stehney. Security quality requirements engineering (square) methodology. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, May 2005.

[15] D. Mellado, E. Fernández-Medina, and M. Piattini. Applying a security requirements engineering process. *Computer Security–ESORICS 2006*, pages 192–206, 2006.

[16] Haralambos Mouratidis and Paolo Giorgini. Secure tropos: a security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, 17(2):285–309, 2007.

[17] G. Silowash, D. Cappelli, A.P. Moore, R. F. Trzeciak, T. J. Shimeall, and L. Flynn. Common sense guide to mitigating insider threats. Technical Report CMU/SEI-2012-TR-012, Software Engineering Institute, Carnegie Mellon, December 2012.

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

[19] A. Van Lamsweerde and Emmanuel Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, 2000.

[20] ESK Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto, Canada, 1995.

[21] Nicola Zannone. *A Requirements Engineering Methodology for Trust, Security, and Privacy*. PhD thesis, University of Trento, Italy, 2007.

Table 8: Axioms for identification of insider threats

Insider Threat to Confidentiality	
T1	**a** $threat(A1, asset\_instance(service\_instance(S,A,P),A,P), confidentiality,\textbf{extreme})\leftarrow asset\_instance(service\_instance(S,A,P),A,P) \wedge sec\_req\_instance(service\_instance(S,A,P), confidentiality, A, P) \wedge permission\_instance(A1, service\_instance(S,A,P), access) \wedge sensitivity\_instance(service\_instance(S,A,P),\textbf{very high}, A, P) \wedge trust\_perm\_instance(A, A1, asset\_instance(service\_instance(S,A,P), access, \textbf{very bad}) \wedge A1 \neq A$ **b** $threat(A1, asset\_instance(service\_instance(S,A,P),A,P), confidentiality,\textbf{extreme})\leftarrow asset\_instance(service\_instance(S,A,P),A,P) \wedge sec\_req\_instance(service\_instance(S,A,P), confidentiality, A, P) \wedge permission\_instance(A1, service\_instance(S,A,P), access) \wedge sensitivity\_instance(service\_instance(S,A,P),\textbf{very high}, A, P) \wedge trust\_perm\_instance(A, A1, asset\_instance(service\_instance(S,A,P), access, \textbf{bad}) \wedge A1 \neq A$ **c** $threat(A1, asset\_instance(service\_instance(S,A,P),A,P), confidentiality,\textbf{extreme})\leftarrow asset\_instance(service\_instance(S,A,P),A,P) \wedge sec\_req\_instance(service\_instance(S,A,P), confidentiality, A, P) \wedge permission\_instance(A1, service\_instance(S,A,P), access) \wedge sensitivity\_instance(service\_instance(S,A,P),\textbf{very high}, A, P) \wedge trust\_perm\_instance(A, A1, asset\_instance(service\_instance(S,A,P), access, \textbf{neutral}) \wedge A1 \neq A$
Insider Threat to Integrity	
T2	**a** $threat(A1, asset\_instance(service\_instance(S,A,P),A,P), integrity,\textbf{extreme})\leftarrow asset\_instance(service\_instance(S,A,P),A,P) \wedge sec\_req\_instance(service\_instance(S,A,P), integrity, A, P) \wedge permission\_instance(A1, service\_instance(S,A,P), modify) \wedge sensitivity\_instance(service\_instance(S,A,P),\textbf{very high}, A, P) \wedge trust\_perm\_instance(A, A1, asset\_instance(service\_instance(S,A,P), modify, \textbf{very bad}) \wedge A1 \neq A$ **b** $threat(A1, asset\_instance(service\_instance(S,A,P),A,P), integrity,\textbf{extreme})\leftarrow asset\_instance(service\_instance(S,A,P),A,P) \wedge sec\_req\_instance(service\_instance(S,A,P), integrity, A, P) \wedge permission\_instance(A1, service\_instance(S,A,P), modify) \wedge sensitivity\_instance(service\_instance(S,A,P),\textbf{very high}, A, P) \wedge trust\_perm\_instance(A, A1, asset\_instance(service\_instance(S,A,P), modify, \textbf{bad}) \wedge A1 \neq A$ **c** $threat(A1, asset\_instance(service\_instance(S,A,P),A,P), integrity,\textbf{extreme})\leftarrow asset\_instance(service\_instance(S,A,P),A,P) \wedge sec\_req\_instance(service\_instance(S,A,P), integrity, A, P) \wedge permission\_instance(A1, service\_instance(S,A,P), modify) \wedge sensitivity\_instance(service\_instance(S,A,P),\textbf{very high}, A, P) \wedge trust\_perm\_instance(A, A1, asset\_instance(service\_instance(S,A,P), modify, \textbf{neutral}) \wedge A1 \neq A$
Insider Threat to Availability	
T3	**a** $threat(A1, asset\_instance(service\_instance(S,A,P),A,P), availability,\textbf{extreme})\leftarrow asset\_instance(service\_instance(S,A,P),A,P) \wedge sec\_req\_instance(service\_instance(S,A,P), availability, A, P) \wedge permission\_instance(A1, service\_instance(S,A,P), manage) \wedge sensitivity\_instance(service\_instance(S,A,P),\textbf{very high}, A, P) \wedge trust\_perm\_instance(A, A1, asset\_instance(service\_instance(S,A,P), manage, \textbf{very bad}) \wedge A1 \neq A$ **b** $threat(A1, asset\_instance(service\_instance(S,A,P),A,P), availability,\textbf{extreme})\leftarrow asset\_instance(service\_instance(S,A,P),A,P) \wedge sec\_req\_instance(service\_instance(S,A,P), availability, A, P) \wedge permission\_instance(A1, service\_instance(S,A,P), manage) \wedge sensitivity\_instance(service\_instance(S,A,P),\textbf{very high}, A, P) \wedge trust\_perm\_instance(A, A1, asset\_instance(service\_instance(S,A,P), manage, \textbf{bad}) \wedge A1 \neq A$ **c** $threat(A1, asset\_instance(service\_instance(S,A,P),A,P), availability,\textbf{extreme})\leftarrow asset\_instance(service\_instance(S,A,P),A,P) \wedge sec\_req\_instance(service\_instance(S,A,P), availability, A, P) \wedge permission\_instance(A1, service\_instance(S,A,P), manage) \wedge sensitivity\_instance(service\_instance(S,A,P),\textbf{very high}, A, P) \wedge trust\_perm\_instance(A, A1, asset\_instance(service\_instance(S,A,P), manage, \textbf{neutral}) \wedge A1 \neq A$