Analyzing and Enforcing Security Mechanisms on Requirements Specifications

Tong Li, Jennifer Horkoff, John Mylopoulos

University of Trento, Trento, Italy
{tong.li,horkoff,jm}@disi.unitn.it

Abstract. [**Context and motivation**] Security mechanisms, such as firewalls and encryption, operationalize security requirements, such as confidentiality and integrity. [**Question/problem**] Although previous work has pointed out that the application of a security mechanism affects system specifications, there is no systematic approach to describe and analyze this impact. [**Principal ideas/results**] In this paper, we investigate more than 40 security mechanisms that are well documented in security pattern repositories in order to better understand what they are and how they function. [**Contribution**] Based on this study, we propose a conceptual model for security mechanisms, and evaluate this model against 20 security mechanisms. Using the conceptual model, we provide a systematic process for analyzing and enforcing security mechanisms on system requirements. We also develop a prototype tool to facilitate the application and evaluation of our approach.

1 Introduction

Dealing with security requirements in the early stages of the system development has become an important topic in Requirements Engineering (RE) and Security research, as software companies have grown tired of spending millions to fix system flaws downstream. Security requirements analysis techniques, such as Misuse Cases [25], Obstacle analysis [26], Secure Tropos [18], involve eliciting security requirements and identifying security mechanisms to fulfill those requirements. Security mechanisms, such as firewalls and encryption, operationalize security requirements, such as confidentiality or integrity. As such, they do not function independently but interact and constrain parts of the system in specific ways. As a result, leveraging a security mechanism not only introduces new requirements to the system, but also inevitably modifies existing system requirements. Viewed as a cross-cutting concern [23], security mechanisms have global impact over the entire system.

Some approaches have claimed that leveraging security mechanisms influences system requirements specifications, which should be iteratively constructed by considering the application of security mechanisms [9, 8]. However, these proposals only focus on new functional requirements that are introduced by a security mechanism and omit their impact on existing functional and non-funtional requirements. In other words, their approaches operationalize security requirements into only individual functional requirements. In addition, there

are neither systematic methods nor supporting tools available for analyzing and enforcing the impact of security mechanisms on system requirements.

We argue that system requirements specifications are not be complete unless they precisely capture such impacts. For example, when applying an access control mechanism to protect a data asset stored in a server, this mechanism imposes global constraints on all functional requirements that involve accessing the server, which should be reflected in the requirements specification in order to correctly develop a secure system. Moreover, the quality of the system functions are affected by the application of security mechanisms, which should be captured and taken into account in order to select the best functional alternatives. For instance, applying the access control mechanism to a specific system function will impair the usability and performance of all related functions provided by the system. Thus, we believe that a security mechanism is not a localized solution that can be independently decided upon over other elements of a requirements specification.

In this paper, we propose to capture and enforce the impact that security mechanisms impose over system requirements in order to completely and correctly account for their integration. Specifically, we investigate, in depth, a collection of security mechanisms that are well documented in security pattern repositories [22, 5], and propose an approach to systematically and semi-automatically generate security-enhanced requirements specifications by analyzing the impact of applying security mechanisms. This work makes the following contributions:

- Presents a conceptual model which characterizes security mechanism from a requirements viewpoint.
- Proposes a systematic way to analyze and enforce the impact of a security mechanism imposed on system requirements. A set of corresponding logic rules are proposed to semi-automate the analysis process.
- Evaluates the expressiveness and effectiveness of our proposal by modeling 20 security mechanisms (selected from [22, 5]) according to the proposed conceptual model and applying the obtained models to a real healthcare network scenario.
- A prototype tool has been developed to support the analysis process.

In the remainder of this paper, we introduce the background of this work (Section 2). We then present an illustrating example used throughout the paper in Section 3. In Section 4, we describe an enriched requirements specification, used as an input to our approach. We then present a conceptual model for security mechanisms in Section 5, along with a systematic process for analyzing the impact of security mechanisms (Section 6). After that we describe the evaluation of our approach in Section 7, and discuss related work in Section 8. In Section 9, we conclude the paper and discuss future work.

2 Background

In this section, we introduce the research baseline for our research.

Requirements Specification Concepts. Our previous work proposed a three-layer requirements analysis framework to analyze requirements, particularly security requirements, in different abstraction layers of Socio-Technical Systems (STS) [14]. This framework offers a holistic approach to analyze security issues in all layers, which takes into account the influences across layers. The requirements modeling language used in that work is based on the core ontology of RE [11], and is further expanded with social concepts that are adopted from *i** [27]. In addition, we use *security goals*, which are specializations of *softgoals*, to capture security requirements in the three-layer framework. Each security goal is specified with *importance*, *security property*, and *asset*, e.g., "*High data confidentiality [Clinical information]*". A security goal is operationalized into a single security mechanism, which is treated as a specialization of a task. Fig. 1 shows a piece of a requirements model that is modeled in our three-layer framework.

In this paper, we specify requirements as in our previous work. In particular, we reuse the concepts: *goal (G)*, *softgoal (SG)*, *task (T)* (i.e., function), *domain assumption (DA)*, and the *refinement (REF)* and *contribution (CON)* relations, while adding a new concept *task constraint (TC)* in order to capture the impact of security mechanisms on existing tasks.

Security Knowledge Sources. With the aim of supporting non-security experts to carry out security requirements analysis and advancing the practical adoption of the analysis, we base our approach on existing security knowledge sources, namely, security patterns. Security patterns provide proven security solutions, through security mechanisms, for known security problems encountered in specific contexts. A number of security pattern repositories have been summarized in literature [5][22][7], which result in more than 100 security patterns in total. A security pattern is specified in a number of sections (depending on the selected pattern template), each of which addresses an aspect of the pattern. An example of the Virtual Private Network (VPN) security pattern is shown in Table 1, which follows the POSA (Pattern-Oriented Software Architecture) template [2] and presents a part of four sections, including *Context*, *Problem*, *Solution*, and *Consequence*.

In this paper, we extract security knowledge from well documented security patterns, specifically the work done by Fernandez et al. [5] and Scandariato et al. [22]. In particular, this paper exclusively focuses on the security mechanism (i.e. the solution) that is provided by each security pattern, while the reason for applying the security mechanism (i.e., the problem) was captured and analyzed in our previous work [15]. As we aim to analyze the impact of security mechanisms on system requirements, we mainly extract the knowledge of security mechanisms from the *Solution* section that specifies the requirements that need to be satisfied by a security mechanism, rather than the *Implementation* section that describes detailed design of a security mechanism.

Modeling and Analyzing Security Patterns. Our previous work proposed to seamlessly integrate security patterns into security requirements analysis by modeling security patterns as contextual goal models, which facilitates the context-based selection among alternative security mechanisms [15]. After

Table 1: Part of the description of the Virtual Private Network pattern [5]

Context: Users scattered in many fixed locations, who need to communicate securely with each other.
Problem: How do we establish a secure channel for the end users of a network so that they can exchange messages through fixed points using an insecure network?
Solution: Protect communications by establishing a cryptographic tunnel between endpoints on one of the layers of the communication protocol.
Consequence: There is some overhead in the encryption process.

choosing the best security pattern, we apply its corresponding security mechanism that is modeled by using *tasks*, *domain assumptions*, and *softgoals*. In this method, the application of a security mechanism involves directly attaching the security mechanism model into the requirements model via refinement and contribution links.

However, this approach does not consider the impact of the mechanism on *existing* functional requirements, including how the impact further affects related non-functional requirements. Capturing and analyzing such impact is a non-trivial task. Take the VPN security mechanism as an example, which is described in the solution section in Table 1. This mechanism requires endpoints to communicate via a cryptographic tunnel, i.e., encrypting the communications. To correctly apply the mechanism, all the functional requirements that communicate confidential information should be constrained by this mechanism, and these requirements are not easy to identify. In addition, as described in the *Consequence*, the VPN mechanism impairs system performance. Thus, all the functional requirements that are constrained by VPN will have a negative influence on system performance, and this influence has to be taken into account when selecting alternative requirements.

In this paper, we propose a method which tackles the above challenges. In particular, we build upon our previous work and create a conceptual model for security mechanisms, which specializes tasks into security tasks, specifies security constraints post by security tasks, and captures the impact of security tasks on non-functional requirements. Based on this model, we are able to systematically analyze the impact of applying security mechanisms on existing requirements.

3 Scenario: The Healthcare Collaborative Network (HCN)

The HCN is a system that enables the exchange of healthcare messages and documents between and within organizations. The essential parts of the HCN include an admin server and a message flow server, which communicate with gateways deployed at both the publisher side and the subscriber side. A full

description of the HCN can be found online[1]. Fig. 1 shows part of the requirements goal model of the HCN, which captures the publisher gateway application, modeled using our existing framework [14]. Note that we assign unique identifiers to each node in the figure in order to facilitate the references in the remaining part of this paper.

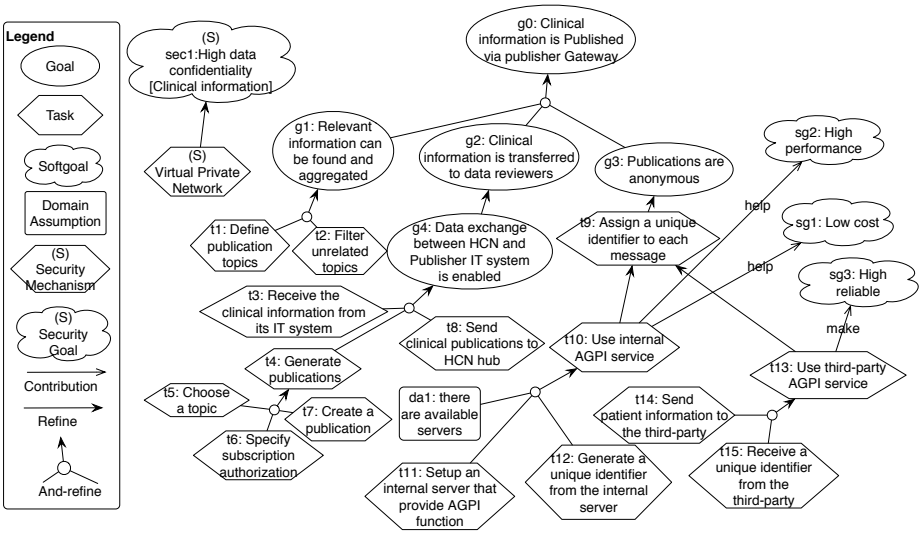

Fig. 1: A snippet of requirements goal model of HCN

4 An Enriched Requirements Specification

In this paper, we use an enriched requirements specification. Such specifications consist of not only goals (G), softgoals (SG), tasks (T), domain assumptions (DA), refinements (RE) and contributions (CON), but also task constraints (TC), which reflect the impact of security mechanisms on tasks. Thus, an enriched requirements specification is defined as a 7-tuple, i.e.,

$$\mathcal{R} = \{G, SG, T, DA, REF, CON, TC\}$$

A task constraint is specified in terms of task invariants and pre/post-conditions. The invariants describe properties that have to be true during the entire execution of the task. The pre/post-conditions describe properties that have to hold before/after the execution of the task. The value of a task constraint can be either a constant (e.g. *user_data*) or a predicate (e.g. *encrypted(user_data)*).

Fig. 1 presents an example of a requirements specification, including all these concepts except for task constraints. Note that the notation of the security mechanism shown in Fig. 1 (task with (S) annotation) is only used as a placeholder, as described in our previous work [14]. This placeholder indicates a security

[1] http://www.redbooks.ibm.com/redbooks/SG246779

mechanism is applied to achieve the security goal. In this work, this notation is replaced by detailed concepts of a security mechanism, as described in Section 5.

Expanded Attributes of Tasks. In order to better analyze the semantics of tasks, we associate each task with three attributes: its *subject*, *object*, and *operation*. For example, as shown in Fig. 2, we detail the selected task with a subject *publisher_gateway_application*, an object *clinical_publications*, and an operation *send*.

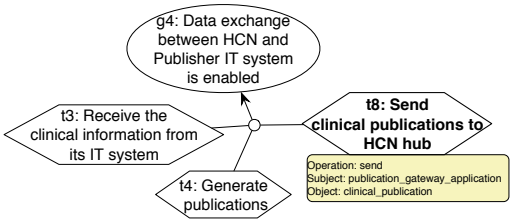

Fig. 2: An example of the enriched requirements elements

Such enriched requirements specifications are treated as the input of the analysis, i.e., each task has to be specified with a subject, an object, and an operation in order to be processed with our approach. During the requirements elicitation phase, there are two ways in which the above detailed information can be collected: firstly, interactively asking users when needed; secondly, automatically extracting the information from textual descriptions of tasks that have been elicited from stakeholders (with manual verification). For the second means, we leverage Nature Language Processing (NLP) as proposed in [13] to identify the roles of sentences, such as subjects, operations, and objects. In particular, we identify the Parts of Speech (POS) for each single word of a requirement statement. Then, we define a set of semantic patterns by using regular expression in order to capture the semantics of each sentence in terms of its subject, operation, and object. This technique has been implemented as part of our prototype tool (Section 7).

5 Modeling Security Mechanisms

In this section, we propose a conceptual model to characterize security mechanisms from a requirements perspective. In particular, a security mechanism is specified in terms of *security tasks*, *assumptions*, *security constraints*, and *quality influences*. As this paper exclusively analyzes the impact of security mechanisms imposed on the requirements specification that has been presented in Section 4, we map the concepts of the security mechanism to the requirements specification concepts as much as possible. In the reminder of this section, we describe each of the concepts that we use to model a security mechanism. An example of the VPN security mechanism is used for illustration, which is shown in Fig. 3.

Security Tasks. A *security task* is a detailed action performed by a system to achieve certain security goals. We define the security task as a specialization of

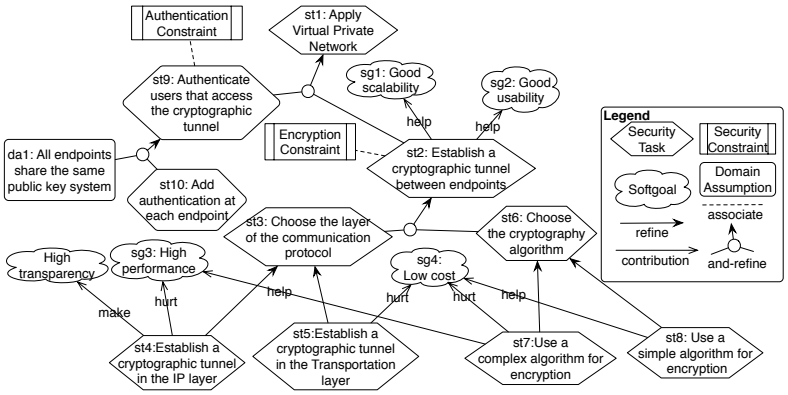

Fig. 3: Modeling security mechanism — virtual private network (VPN)

task, and use T_S to represent the set of security tasks of a security mechanism. Each security task has an additional attribute "*asset*", beyond the 3 attributes of regular tasks that are described in the previous section. This attribute specifies the asset that is protected by a security task, from which we can infer the impact of the security task. As the target asset of a security task depends on the application scenario of the security task, the obtention of this attribute is specified during the analysis process, described in Section 6.

As with all tasks, a composite security task can be decomposed into detailed security tasks, and we define the set of refinement relations between security tasks as REF_S. Note that we use the "root" security task to indicate the overall security mechanism, which can be repeatedly refined till reaching "leaf" security tasks, as shown in Fig. 3.

Assumptions. An *assumption* specifies a expected state of affairs, under which the security mechanism can be applied correctly. Normally, these assumptions are captured during the refinements of security tasks, such as the assumption "*All endpoints share the same public key system*", presented in Fig. 3. We map this concept to *domain assumption*, and use DA_S to represent the set of assumptions made in a security mechanism.

Security Constraints. A security mechanism does not exist independently, but interacts and constrains existing system tasks in order to ensure that security requirements are satisfied. Thus, we explicitly capture such interactions between security tasks and tasks in the requirements model by using security constraints. We use *SC* to present the set of security constraints imposed by a security mechanism.

In this paper, we initially summarize 6 security constraints after investigating more than 40 reusable security mechanisms that are documented in a security pattern textbook [5] and a security pattern repository [22]. The 6 security constraints include *Encryption Constraint*, *Authentication Constraint*, *Permission Constraint*, *Centralization Constraint*, *Protection Constraint*, and *Auditing Constraint*. Each of these security constraints implies that a security task constrains specific tasks which have certain properties. Thus, regarding the meaning of each secu-

Table 2: Security constraint rules

Global impact of security constraints
Rule_1: $constrain(ST,T) \leftarrow has\_operation(T,F) \wedge transfer\_operation(F)$ $\wedge has\_Object(T,O) \wedge protect(ST,O) \wedge has\_constraint(ST, encryption\_constraint)$
Rule_2: $constrain(ST,T) \leftarrow has\_operation(T,F) \wedge (protect(ST,F)$ $\vee (access\_operation(F) \wedge has\_object(T,O) \wedge protect(ST,O)))$ $\wedge has\_constraint(ST, authentication\_constraint)$
Rule_3: $constrain(ST,T) \leftarrow has\_operation(T,F) \wedge (protect(ST,F)$ $\vee (access\_operation(F) \wedge has\_object(T,O) \wedge protect(ST,O)))$ $\wedge has\_constraint(ST, authorization\_constraint)$
Rule_4: $constrain(ST,T) \leftarrow has\_operation(T,F) \wedge protect(ST,F)$ $\wedge has\_constraint(ST, centralization\_constraint)$
Rule_5: $constrain(ST,T) \leftarrow has\_operation(T,F) \wedge access\_operation(F)$ $\wedge has\_Object(T,O) \wedge protect(ST,O) \wedge has\_constraint(ST, protection\_constraint)$
Rule_6: $constrain(ST,T) \leftarrow (has\_function(T,F) \wedge protect(ST,F))$ $\vee (has\_Object(T,O) \wedge protect(ST,O)) \wedge has\_constraint(ST, auditing\_constraint)$

rity constraint, we define security constraint rules for each particular security constraint to identify tasks that are constrained by a security task. The full list of security constraint rules are shown in Table 2. Take the *Rule 1* as an example: if a security task *ST* has an *encryption_constraint*, which targets the asset O, and there is a task T that has an operation F, which *transfers* the asset O, then the task T is constrained by the security task *ST*. Once having a list of security constraints, we need to go through each security task modeled before to identify whether it imposes certain security constraint. For example, as shown in Fig. 3, we identify that the security tasks *st2* and *st9* impose the *Encryption Constraint* and *Authentication Constraint*, respectively.

The proposed security constraints are not intended to be complete, but provide good coverage when considering the content of the 40 investigated security patterns. Additional constraints, together with their corresponding constraint rules (e.g. Table 2), can be incrementally integrated into our work.

Quality influences. Each security task not only changes functions of a system, but may also influence the qualities of the system, either positively or negatively. We use a set of *contribution* links to capture such quality influences, which are represented as CON_S. A contribution link is a triple, which specifies the influence imposed by a security task over system related quality (captured as a softgoal). We define the set of softgoals affected by a security mechanism as SG_S. Thus, the quality influences are defined as:

$$CON_S \subseteq T_S \times \{make, help, hurt, break\} \times SG_S$$

For example, in Fig. 3, the security task "*Establish a cryptographic tunnel in the IP layer*" *makes* the softgoal "*High transparency*", while *hurts* another softgoal "*High performance*".

6 Analyzing the Impact of Security Mechanisms

In this section, we propose a systematic process to analyze and enforce the impact security mechanisms impose on the existing system requirements specification. We take the enriched requirements specification $\mathcal{R}$ and the to-be-applied security mechanism specification $\mathcal{M}$ as the input of our analysis, i.e.,

Input: $\mathcal{R} = \{G, SG, T, DA, REF, CON, TC\}$, $\mathcal{M} = \{T_S, REF_S, DA_S, SC, SG_S, CON_S\}$

After systematically analyzing the impact of the security mechanism (Fig. 4), our approach will generate an updated requirements specification, $\mathcal{R}'$, which reflects all the impacts of the security mechanisms imposed on the requirements specification, i.e.,

$$\text{Output: } \mathcal{R}' = \{G', SG', T', DA', REF', CON', TC'\}$$

We illustrate the analysis process by analyzing the impact of the VPN mechanism (Fig. 3) imposed on the piece of requirements specification of the HCN scenario (Fig. 1). It is worth noting that if there are multiple security mechanisms need to be applied, all of them will be analyzed iteratively using the same approach.

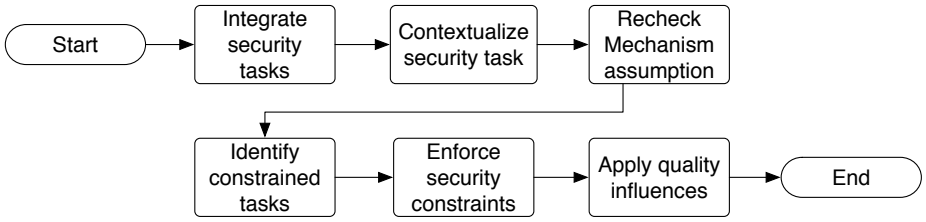

Fig. 4: The process for analyzing impact of security mechanisms

Integrate Security Tasks. All security tasks, as a specialization of tasks, are directly incorporated into the initial requirements specification, as well as the refinements relations among them (if they exist). As such, the integration is defined as follows:

$$T = T \cup T_S \text{ , } REF = REF \cup REF_S$$

As a security mechanism is applied to operationalize a security goal, the *root* security task of the security mechanism will replace the placeholder described in Fig. 1, and is directly linked to the security goal. In the illustrating example, the result of integrating security tasks of the VPN mechanism to the requirements specification is shown in the right part of Fig. 5 (*st1-st10*).

Contextualize Security Tasks. Once security tasks are integrated into the requirements and linked to a particular security goal, the target assets of security tasks should be determined in order to support the identification of constrained tasks in a later step. Each security goal in the requirements specification has already been specified an asset, such as the security goal *sec1* is specified with an asset "*clinical_information*" (Fig. 5). Thus, the security tasks that are applied to satisfy a security goal will inherit the asset from that security goal. In the illustrating example (Fig. 5), all the applied security tasks have the asset "*clinical_information*", automatically derived from the security goal *sec1*.

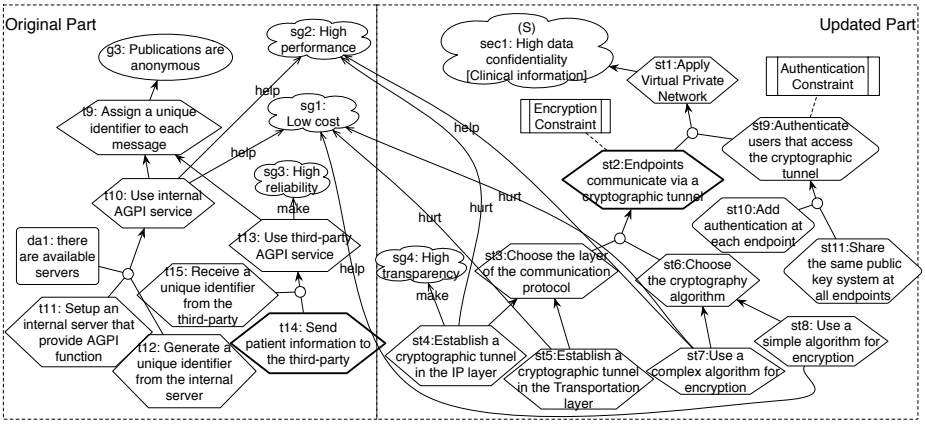

Fig. 5: Impact of the application of VPN (part)

Recheck Assumptions. When applying a security mechanism to a system within a particular domain, assumptions made in the mechanism should be further checked about whether or not it is still an assumption in the domain. Thus, a heuristic question can be asked, "Is the assumed phenomenon inside the boundary of system design now?" If so, we need to replace this assumption with a security task which "realizes" the assumption, and then add this security task to the set of tasks, i.e.,

$$T = T \cup \{a | \forall a \in DA_S, inside\_design\_boundary(a)\}$$

In this case, the newly added security tasks should be appropriately performed to ensure that the security mechanism is executed correctly. If the answer to the question is "No", the properties in the assumption keep being assumed to be held, and we add the assumption to the set of domain assumptions, i.e.,

$$DA = DA \cup \{a | \forall a \in DA_S, outside\_design\_boundary(a)\}$$

In our example, the assumption of the VPN mechanism "*All endpoints share the same public key system*" is determined to be inside the system design boundary. So we create a security task regarding this assumption (i.e., the *st11* in Fig. 5), and add this security task to the set of tasks.

Identify Constrained Tasks. After security tasks have been contextualized with the asset information, we now apply the security constraint rules (Table 2) to automatically identify the interactions between security tasks in the security mechanism and tasks in the requirements specification, i.e., identifying which tasks are constrained a security task.

During the above impact identification, we are concerned about not only the information derived from the two specifications (i.e., $\mathcal{R}$ and $\mathcal{M}$), but also additional domain knowledge models, such as data schemes (Fig. 6 (a)) and semantic hierarchies of words (Fig. 6 (b)). These models provide auxiliary rules to facilitate the analysis, e.g., the following rules:

$$\textit{Rule\_7: } protect(ST, A2) \leftarrow protect(ST, A1) \wedge part\_of(A1, A2)$$

$$\textit{Rule\_8: } transfer\_opertiona(O) \leftarrow send\_operation(O)$$

Rule 7 indicates that if an asset needs to be protected, all the parts of this asset

also should be protected. Rule 8 indicates that if an operation is of the type of "send", then it is also of the type of "transfer".

In our example, we use Rule 1 to infer, and identify three tasks {*t3, t8, t14*} (Fig. 1), which are constrained by the security task *st2*. Due to space limitation, Fig. 5 only represents part of the original requirements model that is related to *t14*.

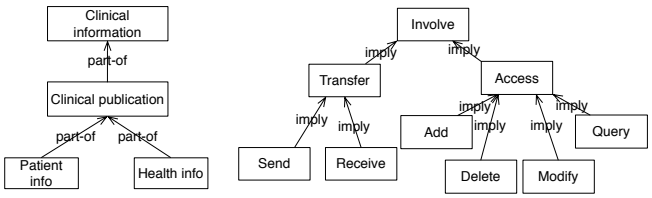

Fig. 6: Examples of knowledge models

Enforce Security Constraints. After identifying all tasks that are constrained, we further enforce security constraints on those tasks. In particular, we propose specific enforcement measures for each of the 6 security constraints according to their meanings, which are detailed in Table 3. In this table, we first present the impact introduced by each security constraint. After that we describe the concrete enforcement measures, which are either adding task constraints or replacing tasks. For example, the *Encryption Constraint* adds a new pre-condition to the constrained task, the *Protection Constraint* adds a new invariant to the constrained task, and the Auditing *Constraint* adds a new post-condition to the constrained task. Apart from imposing task constraints, the *Centralization Constraint* replaces the constrained task with the corresponding security task. In this case, all the refinement relations that were linked to the constrained task are now redirected to the security task, and then the constrained task is removed.

According to the proposed enforcement measures, in our example, we enforce the encryption constraint on the constrained task $t14$ (Fig. 5, i.e., adding a new pre-condition $performed(st2)$ to this task.

Apply Quality Influences. Many requirements analysis techniques rely on qualities, which are normally captured as non-functional requirements (NFRs), to select alternative requirements [10]. Due to the interactions between security tasks and tasks, the quality influences introduced by security tasks may affect system requirements decisions, which need to be re-evaluated.

As the first step of applying quality influences, we correlate the softgoals in SG_S with the softgoals in SG, i.e. checking whether they are the same softgoals. As the same concept may be presented by different terms in different ways, this correlation analysis may require additional techniques, such as the Repertory Grid Technique (RGT) [19]. In the illustrating example, SG_S of the VPN mechanism involves several softgoals among which "*Low cost*" and "*High performance*" have been correlated with softgoals in SG (in this particular case, the correlated softgoals have the same contents). For the softgoals in SG_S that are not correlated, the analyst needs to re-evaluate stakeholders' non-functional

Table 3: Enforcement measures for the 6 security constraints

Security Constraints	**impact**	**Enforcement**
Encryption Constraint	the encryption security task should be done before the constrained task.	*add(performed(st), t.precondition)*
Authentication Constraint	the authentication security task should be done before the constrained task.	*add(performed(st), t.precondition)*
Permission Constraint	the authorization security task should be done before the constrained task.	*add(performed(st), t.precondition)*
Centralization Constraint	the constrained task is replaced by the centralized security task.	*replace(t, st)*
Protection Constraint	the protection security task should be enforced to cover the whole execution period of the constrained task.	*add(cover_by(st), t.invariant)*
Auditing Constraint	the auditing security function should be done after the execution of the constrained task.	*add(need_to_perform(st), t.postcondition)*

Remark: the *st* indicates the corresponding security task of a security constraint, while the *t* stands for the constrained task.

requirements to decide whether to include these softgoals. In our example, the uncorrelated softgoal "*High traceability*" is evaluated, and a decision is made to add it to the *SG*, shown in Fig. 5. This integration is defined below,

$$SG = SG \cup \{sg | \forall sg \in SG_S, uncorrelated(sg) \wedge decide\_include(sg)\}$$

However, the other uncorrelated softgoals, such as "*High usability*", are evaluated and are determined to not fit in with the current scenario. Once the above correlated softgoals and newly added softgoals are determined, all their corresponding contribution links in CON_S will be integrated into the requirements specification, i.e.,

$$CON = CON \cup \{contribute(st, inf, sg) | \forall contribute(st, inf, sg) \in CON_S, \exists sg \in SG\}$$

After correlating softgoals, we analyze the quality influences of a security task to its constrained tasks. Specifically, if a security task constrains a task, then all the quality influences introduced by this security task should be taken into account when evaluating the constrained task, especially if the constrained task is part of a requirements alternative. In the example (Fig. 5), since *t14* is constrained by *st2*, the correct execution of *t14* requires the appropriate interactions with *st2*. Thus, when evaluating the requirements alternatives that involve *t14*, such as the alternative tasks *{t11,t12}* vs. *{t14,t15}*, the influences *st2* imposed on the qualities (i.e., *sg1, sg2, sg4*) have to be taken into consideration.

7 Evaluation

Evaluate Expressiveness. We apply the proposed conceptual model to 20 security mechanisms, which are specified as reusable security solutions in the security pattern textbook [5]. The statistics of applying the conceptual model to the 20 security mechanisms are presented in Table 4. The result of this evaluation shows that the 6 security constraints defined in this paper are enough to

Table 4: Statistics of applying the conceptual model to 20 security mechanisms

	Security Task	Assumption	Security Constraint	Quality Influence
Total	89	15	27	148
Average	4.45	0.75	1.35	7.4

capture the semantics of these security mechanisms, and some security mechanisms impose more than one security constraint. On average each mechanism has more than 4 security tasks, which implies that security mechanisms are normally described at high abstraction level and can be further refined into detailed tasks. Moreover, the large number of quality influences further justify that security mechanisms can heavily affect the quality of systems, the impact of which should be carefully inspected. On the whole, by applying the conceptual model, a single security mechanism has around 14 nodes on average. Thus, the conceptual model is scalable to model a larger number of security mechanisms and include them into the repository.

Evaluate Effectiveness. We apply the proposed analysis approach to the full requirements model that we have built for the HCN scenario, which contains 23 goals, 8 softgoals, 67 tasks, and 75 refinement links. In particular, we analyze the impact of the VPN mechanism (Fig. 3), which has 9 security tasks, 1 assumption, 2 security constraints, and 8 quality influences. The application of this mechanism identifies 12 constrained tasks, each of which has applied 2 task constraints and 3 quality influences. This evaluation shows that our approach is able to identify and enforce the impact of security mechanisms, and it is scalable to a medium-size requirements model.

Tool Support. We have developed a prototype tool to support the evaluation and application of our approach. This prototype is built on top of our previous security requirements analysis tool MUSER [16]. Apart from the features provided by MUSER, e.g., graphically modeling requirements goal models, this prototype tool helps analysts to model security mechanisms and analyze their impact on requirements models. Specifically, the tool can infer the tasks that are constrained by specific security tasks according to the proposed rules. If certain information is missing during the reasoning process, the tool will interactively ask users to provide relevant information. Finally, as mentioned in Section 4, we leverage another tool to facilitate generating enriched requirements specification, which automatically extracts the *subject*, *object*, and *operation* from the description of a task by using NLP techniques [13].

8 Related Work

The interaction between requirements and architecture was first emphasized by Nuseibeh in [20], where he proposes a twin peaks model to show these interactions at an abstract level. Heyman et al. [9] and Okubo et al. [21] specialize the twin peaks model in the security area, respectively. They all outline a constructive process for co-developing secure software architectures and security

requirements, but do not consider the impact secure architectures impose on other non-security requirements. In addition, none of these approaches has formalized the interactions between the twin peaks, and there is no tool developed to support the analysis process.

In Goal-Oriented Requirements Engineering (GORE), stakeholder's requirements, i.e., goals and softgoals should be operationalized into specific functions. As summarized by Dalpiaz et al. [3], there are several types of operationalization among existing GORE approaches, namely: functional requirements operationalization, qualitative operationalization, adaptation requirements operationalization, and behavior operationalization. Most of the existing work about security requirements operationalization falls into the first category, i.e., operationalizing security requirements into particular functions [17, 8, 14]. However, in this paper, we argue that any single category summarized above is not enough to characterize the operationalization of security requirements. Instead, our proposal aims to provide a new category of requirements operationalization, which focuses on capturing various changes on existing requirements specification.

Apart from the type of requirements operationalization, the means of doing the operationalization is also an essential step of the analysis. Letier and Lamsweerde have proposed to leverage operationalization patterns to guide the operationalization analysis [12], while Alrajeh et al. leverage machine learning techniques to operationalize goals [1]. As these approaches help to guarantee the correctness of the obtained operational specification, they can complement our work during the step of enforcing security constraints, specifically, validating the enforcement rules.

Security, as a cross-cutting concern, has been investigated in an aspect-oriented manner. Gunawan et al. model both systems functional designs and security mechanisms by using the collaboration-oriented behavior model, and propose to treat each security mechanism as a security aspect that can be inserted into different places of the system design [6]. Sousa et al. adapt the NFR framework to support aspect-oriented analysis [4]. Specifically, they illustrate their approach with a security requirements example, as they treat security requirements as a NFR. However, the above approaches do not consider the quality influences imposed by security mechanisms.

The impact of security mechanisms have been enforced by using model transformation techniques. Shiroma et al. focus on applying security mechanisms onto UML class diagrams [24]. They automatically enforce the security mechanism by defining transformation rules in ATLAS transformation language. However, this work focuses on the design phase and does not consider the impact on the system requirements. Yu et al. use i* constructs to model the context, problem, and solution of a security pattern, and automate the problem matching and application of the security solution by using ATL [28]. However, their approach highly depends on the semantics of the constructs of i*, such as dependencies and roles, and cannot be generalized for all security mechanisms, such as encryption.

9 Conclusions and Future Work

In this paper, we propose a conceptual model, which characterizes security mechanisms as security tasks, assumptions, security constraints, and quality influences. Using this conceptual model, we provide a systematic way to analyze and enforce the impact that security mechanisms impose over the system requirements. By defining related reasoning rules and implementing a prototype tool, the proposed analysis can be semi-automated. Finally, we evaluate the expressiveness of our conceptual model against 20 security mechanisms documented in existing security pattern repositories, and further evaluate the effectiveness of the analysis approach using a HCN scenario.

In the future, we want to generalize our approach to other goal-oriented security analysis approaches, such as Secure Tropos, KAOS. To this end, the proposed conceptual model of security mechanisms should be appropriately mapped to other types of goal-oriented requirements specifications, and the analysis process should also be adjusted accordingly. Another branch of the future work involves generalizing our approach to analyze and enforce the impact of all kinds of mechanisms (e.g., safety mechanisms, performance mechanisms, etc.) on the requirements specifications.

Apart from the above generalization of this work, we aim to collect more empirical evidence of the effectiveness of our solution, based on which we can further improve the approach. Firstly, beyond the 20 security mechanisms that have been specified in our conceptual model, we plan to analyze more security mechanisms to further check the coverage of the 6 security constraints proposed in this paper. Secondly, larger scale case studies will be done to better evaluate the effectiveness of our approach. Thirdly, we want to involve practitioners into the evaluation of the approach via controlled experiments to evaluate the potential of the practical adoption of our approach.

Acknowledgements This work was supported in part by ERC advanced grant 267856, titled "Lucretius: Foundations for Software Evolution".

References

1. D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel. Learning operational requirements from goal models. In *Proceedings of the 31st International Conference on Software Engineering*, pages 265–275, 2009.
2. F. Buschmann, K. Henney, and D. Schimdt. *Pattern-oriented Software Architecture: On Patterns and Pattern Language*, volume 5. John Wiley & Sons, 2007.
3. F. Dalpiaz, V. E. S. Souza, and J. Mylopoulos. The many faces of operationalization in goal-oriented requirements engineering. In *Proceedings of the Tenth Asia-Pacific Conference on Conceptual Modelling-Volume 154*, pages 3–7, 2014.
4. G. M. C. de Sousa, I. G. da Silva, and J. B. de Castro. Adapting the nfr framework to aspect-oriented requirements engineering. In *Proceeding of XVII Brazilian Symposium on Software Engineering*, pages 83–98, 2003.
5. E. Fernandez-Buglioni. *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons, 2013.
6. L. A. Gunawan, P. Herrmann, and F. A. Kraemer. Towards the integration of security aspects into system development using collaboration-oriented models. In *Security Technology*, pages 72–85. Springer, 2009.

7. M. Hafiz, P. Adamczyk, and R. E. Johnson. Organizing security patterns. *IEEE Software*, 24(4):52–60, 2007.
8. C. B. Haley, R. Laney, J. D. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. *Software Engineering, IEEE Transactions on*, 34(1):133–153, 2008.
9. T. Heyman, K. Yskout, R. Scandariato, H. Schmidt, and Y. Yu. The security twin peaks. In *Engineering Secure Software and Systems*, pages 167–180. Springer, 2011.
10. J. Horkoff and E. Yu. Comparison and evaluation of goal-oriented satisfaction analysis techniques. *Requirements Engineering*, 18(3):199–222, 2013.
11. I. J. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the core ontology and problem in requirements engineering. In *International Requirements Engineering, 2008. RE'08. 16th IEEE*, pages 71–80. IEEE, 2008.
12. E. Letier and A. van Lamsweerde. Deriving operational software specifications from system goals. In *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 119–128, 2002.
13. J.-B. Li, T. Li, and L. Liu. Chinese requirements analysis based on class diagram semantics. *Acta Electronica Sinica*, page S1, 2011.
14. T. Li and J. Horkoff. Dealing with security requirements for socio-technical systems: A holistic approach. In *CAiSE'14*, 2014.
15. T. Li, J. Horkoff, and J. Mylopoulos. Integrating security patterns with security requirements analysis using contextual goal models. In *the 7th IFIP WG 8.1 working conference on the Practice of Enterprise Modelling (PoEM'14)*, 2014.
16. T. Li, J. Horkoff, and J. Mylopoulos. A prototype tool for modeling and analyzing security requirements from a holistic viewpoint. In *the CAiSE'14 Forum at the 26th International Conference on Advanced Information Systems Engineering*, 2014.
17. H. Mouratidis and P. Giorgini. A natural extension of tropos methodology for modelling security. In *Proc. of the Agent Oriented Methodologies Workshop (OOPSLA 2002)*, Seattle-USA,, 2002. Citeseer.
18. H. Mouratidis and P. Giorgini. Secure tropos: a security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, 17(02):285–309, 2007.
19. N. Niu and S. Easterbrook. So, you think you know others' goals? a repertory grid study. *Software, IEEE*, 24(2):53–61, 2007.
20. B. Nuseibeh. Weaving together requirements and architectures. *Computer*, 34(3):115–119, 2001.
21. T. Okubo, H. Kaiya, and N. Yoshioka. Mutual refinement of security requirements and architecture using twin peaks model. In *Computer Software and Applications Conference Workshops (COMPSACW)*, pages 367–372. IEEE, 2012.
22. R. Scandariato, K. Yskout, T. Heyman, and W. Joosen. Architecting software with security patterns. Technical report, KU Leuven, 2008.
23. V. Shah and F. Hill. An aspect-oriented security framework. In *DARPA Information Survivability Conference and Exposition, 2003.*, volume 2, pages 143–145. IEEE, 2003.
24. Y. Shiroma, H. Washizaki, Y. Fukazawa, A. Kubo, and N. Yoshioka. Model-driven security patterns application based on dependences among patterns. In *International Conference on Availability, Reliability, and Security, 2010*, pages 555–559, Feb 2010.
25. G. Sindre and A. L. Opdahl. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44, 2005.
26. A. Van Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, Oct 2000.
27. E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. pages 226–235. IEEE Computer Soc. Press, 1997.
28. Y. Yu, H. Kaiya, H. Washizaki, Y. Xiong, Z. Hu, and N. Yoshioka. Enforcing a security pattern in stakeholder goal models. In *Proceedings of the 4th ACM Workshop on Quality of Protection*, pages 9–14, 2008.