Security Requirements Engineering: State of the Art and Practice and Challenges

Golnaz Elahi

1 Introduction

Security is a property of the system which remains dependable in the face of malice, error, or mischance [3]. In scope of information system, security consists of seven states: confidentiality, integrity, availability, authenticity, accountability, non-repudiation and reliability as defined in ISO/IEC 13335 standard [38]. Introduction of such security goals stem from potential adversaries that attempt to compromise the system. In security terminology, an attacker performs intentional unwarranted actions to break the security of a system by exploiting a vulnerability [71]. The concept of vulnerability is considered as a property of the system or its environment that in conjunction with an attack can lead to the security failure [3]. Assets are another concept in security literature defined as any thing valuable in an organization [38] and the subjects of the attacks [71].

Security requirements engineering frameworks derive security requirements using these security-specific concepts, borrowed from security engineering paradigm. In addition to protection purposes, security requirements are derived from analysis of interactions and dependencies of social actors in the organizational contexts [51, 50]. Such considerations to extract security requirements bring other related issues such as trust among actors, ownership, permission and delegation to security requirements modeling and analysis [23].

Sections 2 and 3 of this paper survey existing security-specific modeling approaches and security requirements framework. It is of interest to study how deriving security requirements based on different security concepts results in different types of security requirements and security design solutions. Existing security-specific modeling and analysis techniques consider different definitions for security requirements resulting from different factors for deriving security requirements.

Surveying security-related modeling notations and security requirements frameworks reveals challenges for developing secure software systems. First, existing security requirements frameworks do not consider potential conflicts between security and other functional and non-functional requirements. In current practice, the interaction of security requirements with other design objectives and goals of stakeholders are not analyzed, if any security requirement is gathered at all. Few research contributions address security trade-off analysis explicitly which are studied in section 4.

Second challenge stems from the tight relationship between architecture design and security requirements and impact of security mechanisms on other requirements. This requires the requirements engineers take the alternative security solutions into consideration to extract security requirements and resolve the security trade-offs. This requires that security requirements framework expand the analysis from the problem space to the solution space as well.

Finally, a major obstacle toward secure software development is the lack of security knowledge and expertise among ordinary software developers. To fill this gap, a major portion of research and practical development in security software engineering is dedicated to develop security knowledge repositories, patterns, security taxonomies, ontologies, and knowledge bases. This paper surveys security knowledge management solutions in section 5. Finally, section 6 gives a conclusion and suggests future research proposals to tackle the current challenges.

2 Modeling Notations for Security

Models capture an abstraction of certain types of information and descriptions, communicate the information, make implicit information explicit, and act as a repository for knowledge and rationale. Conceptual modeling notations are a major area of study in software engineering for facilitating requirements extraction, management, analysis, and visualization. In addition, models can be used for system and architecture design, analysis, verification, and visualization.

Conceptual modeling notations for software engineering practices express different concepts to serve different purposes. However, security issues involve special aspects that traditional software engineering notations do not consider. For example, a general behavior modeling notation expresses interactions of entities in the system without considering the harmful behavior of an external adversary. Thus, the models do not circulate the impacts of the malicious behavior of the adversary on requirements, design, and architecture to the next phases of development.

To model specific security aspects such as attackers and their malicious behavior (attacks), vulnerabilities, assets, and countermeasures several security modeling notations have been developed. We have identified two main categories of notations: one focuses on the attacks and vulnerabilities, and one captures countermeasures and security requirements. We call the former *black side* notations and the latter *white side* modeling approaches. It needs to be mentioned that the categories are not disjoint, and some of the modeling notations may provide conceptual foundation to model both white and black side concepts.

2.1 Black Side: Vulnerability and Attack Modeling

as discussed by Sindre and Opdahl [48], graphical models become much clearer if the distinction between malicious and non-malicious elements is made explicit and the malicious actions are visually distinguished from the legitimate ones. In particular, Sindre and Opdahl show that the use of inverted elements strongly draws the attention to dependability aspects early on for those who discuss the models. The notion of misuse cases [75], abuse cases [55], and malicious actors in the i* Framework [51, 50, 17] are examples of approaches that invert icons to indicate the functionality not wanted in the system.

A misuse case is a use case from the point of view of a hostile actor which poses a threat to the system. Countermeasures for mitigating the threats are expressed as *security use cases* that mitigate misuse cases. Therefore, use case relationships are extended by adding a *prevent* relationship to the misuse case models for relating security use cases to misuse cases. Sindre and Opdahl [75] claim that looking at systems from a misuser perspective increases the chance of discovering threats that would otherwise be ignored.

The other similar notion based on the conventional UML use case modeling is the notion of abuse case [55]. While use cases are abstract episodes of useful and desired interaction between a system and its environment, abuse cases are specification of types of interactions that their results are harm to system. An abuse case describes the abuse of privilege used to complete the abuse case.

Liu et al. [51] analyze attackers, vulnerabilities in actors dependency networks, countermeasure, and access control using the i* modeling framework [81]. Using the approach in [51], all actors are assumed potential attacker, and inherit capabilities, intentions, and social relationships of the corresponding legitimate actor. The actors in the basic dependency model are substituted with their corresponding attackers, and then the impact of the attack to the dependency relationship is traced to the dependency network of other actors. In another use of i* modeling for security analysis [17], inverted i* elements are employed for distinguishing the malicious actors, goals, and tasks.

Another major research efforts are dedicated to developing modeling notations for specifying attacks such as Attack Tree [70], Fault Tree [79], Attack Nets [56], and Attack Graph [65]. Schneier was first to associate the term "Attack Tree" with the use of tree style models for expressing different ways in which a system can be attacked [70]. The Attack Tree is suggested as a formal and methodical way for describing security of the system based on varying attacks. The root node of an attack tree is the goal of the attack and different ways to achieve that goal are leaf nodes, and a node can be decomposed by AND/OR relations. Schneier asserts that designers benefit from understanding all different ways in which a system can be attacked, who the attackers are and what their abilities, motivations, and goals are for selecting proper countermeasures for those attacks.

Fault Tree Analysis (FTA) [79] is one of the most commonly-used techniques in reliability engineering. It determines various combinations of hardware and software failures and human errors To develop the Fault Tree model, system failures are identified and each failure is analyzed on its own Fault Tree. The Fault Tree is refined into more detail events and end up with un-develop/external events, called leaf events. The logical relationship is represented by several types of logical gates such as AND, OR, XOR, PRIORITY, etc. Fault Trees have been used for analysis of failure conditions of complex software systems. For example, Helmer et al. [33] employs Fault Trees for requirements identification and analysis of Intrusion Detection Systems (IDS).

McDermott [56] proposes Attack Nets method for modeling attacks as a Petri Nets with a set of places representing interesting states or modes of the security relevant entities of the system. Using Attack Net modeling approach, attack steps and attacker actions can be expressed. The Attack Nets has a set of transitions that represent input events, commands, or data that cause one or more security relevant entities to change state. The Attack Nets has a set of tokens, which move from place to place along the directed arcs to indicate the progress or concurrency of the attack. The integration of software Fault Tree Analysis (to describe intrusions) and Colored Petri Nets (CPNs) (to specify design) of an IDS is introduced in [32].

Phillips et al. [65] introduced Attack Graphs to analyze vulnerabilities in computer networks. Attack Graphs provide a method for modeling attacks and relating them to the machines in a network and to the attackers. The Attack Graph model is developed based on attack templates, attack profiles, and network configurations. Attack templates describe generic steps in known attacks and conditions which must be hold which are matched with the network configuration and attacker profile to generate the Attack Graph. An Attack Graphs is an attack template instantiated with particular attackers/users and machines. Thereby, one can analyze the Attack Graph by identifying the attack paths that are most likely to succeed.

Sheyner et al. [73] argue that constructing Attack Graphs is a crucial part of vulnerability analysis of a network. Since constructing Attack Graphs by hand is tedious and error-prone, they suggest an automated method for generating and analyzing Attack Graphs. Gupta et al. [27] found the informal attack graph modeling approach useful for iterative system design. In this modeling approach, the system's purpose, potential attacker goals, the path to successful attack, and trust boundaries of the system are modeled and documented.

CORAS framework [15] provides a modeling approach inspired by UML for expressing assets, risks targeting the assets, vulnerabilities, and security solutions. The modeling notation has been improved over time, and [10] introduces new elements for expressing unwanted incidents, risks, accidental and deliberate threats, and stakeholders. The threat models relate a threat to vulnerability and risks are consequently imposed. Risks are related to unwanted incidents, and incidents are connected to the target assets. Finally, treatments can be related to vulnerabilities and risks.

2.2 White Side: Security Requirements and Countermeasures Modeling

Another major group of contributions to conceptual modeling notations for security-specific aspects express the protection mechanisms against the malicious behavior. Although attacks are considered in many of these conceptual modeling languages, these types of contributions mainly focus on modeling security relevant information, security countermeasures, and security requirements. Since security requirements modeling and analysis frameworks are separately studied in section 3, this part is limited to related conceptual modeling notations that capture security specific concepts.

UMLsec [43] is an extension to UML that allows expressing security relevant information within UML diagrams. The main uses of such approach are first, to encapsulate knowledge and make it available to developers in form of a widely-used design notation, and secondly, to provide formal evaluation to check if the constraints associated with the UMLsec stereotypes are fulfilled in a given specification. The extensions are suggested in form of UML stereotypes, tags, and constraints that can be used in various UML diagrams such as activity diagrams, statecharts, and sequence diagrams. The stereotypes and tags encapsulate the knowledge of recurring security requirements of distributed object-oriented systems, such as *secrecy*, *fair exchange*, and *secure communication link*. By assigning a stereotype or tag to part of a model and retrieving the threats, the behavior of the subsystem can be analyzed to check the impacts of the threat and the security of the system with the execution of the threat.

SecureUML [52] is another UML-based modeling language for the model-driven development of secure, distributed systems based on the Unified Modeling Language (UML). SecureUML takes advantage of Role-Based Access Control (RBAC) for specifying authorization constraints by defining a vocabulary for annotating UML-based models with information relevant to access control.

Mouratidis et al. [61] introduce extensions to Tropos methodology [24] for incorporating security concerns into the agent-oriented development process. In this approach, security constraints, secure dependencies, threats, and security goals, tasks, and resources are introduced and added to the Tropos modeling notation. To model these concepts, Tropos modeling notation [11] is used and extended. In this approach, secure entities are tagged with an "s" to indicate those tasks, goals, and softgoals are security-related. Security requirements are dealt as constraints on the functionalities. The assignment of a security constraint to a goal is indicated using a constraint link with a *restricts* tag. Threats represent circumstances that have the potential to cause loss or problems that put the security features in danger. To model threats a new modeling constructs is added to the i* modeling notation as part of the secure Tropos.

In a related work in [59], modeling security capabilities is added to the secure Tropos framework. In a recent work, Matulevicius et al. [54] adapts Secure Tropos proposal for security risk management. In this enhanced framework, the i* beliefs are used for modeling vulnerabilities. In this way, vulnerabilities are treated as attackers' assumptions and believes about system weaknesses. Vulnerabilities are located inside the boundary of actors and the positive impacts of vulnerabilities are related to the attack model. In section 3, the security requirement engineering stages of secure Tropos will be described.

2.3 Discussion

Existing modeling notations for security address different security concepts and take different viewpoints to security. Each modeling approach is able to express certain aspects and lacks conceptual modeling constructs to represent some other. In this part, limitations and capabilities of the methods are studied and compared.

2.3.1 Comparison

Table 1 compares existing modeling notations based on the concepts they express and usage of the models. In some of the notations for modeling attacks and security mechanisms, attacks are modeled in a similar way that security mechanisms are modeled, while the goals and means of the attacker and defender are different. Contributions such as [43] express the need to joint execution of the system with presence of attacks and countermeasures to check if the security mechanisms successfully protect the system against the attacks.

Attack modeling notations mostly focus on decomposing the attacker's goal into a set of actions [70] or required steps and conditions to mount an attack [56, 65]. Several requirements modeling notation capture unwanted behavior of system actors [51, 75, 55]. The concept of vulnerability is considered in few of the contributions. The CORAS framework [15] provides explicit modeling constructs to express weaknesses in the system that may lead to threat scenario. The improved Secure Tropos [54] models vulneerability as i* beliefs. Mayer et al. extended the i* modeling notation by adding two new modeling elements, threats and vulnerabilities as well. Security countermeasures are treated as an general element of the model, and in some approaches, distinguished from other elements of model by an "s" [61].

2.3.2 Limitations of Existing Modeling Approaches

To our knowledge, the efficacy of inverted graphical elements for modeling malicious behavior is not evaluated in empirical studies involving experiments with human subjects. However, several conceptual modeling notations use the notion of inverted elements. The notion of misuse case cannot express due to what vulnerabilities the misuse threatens use cases, why a misuser attacks the system, and what is the impact of a security use case and a misuse case on other use cases. To fill of these gaps, Rostad [68] suggests extending misuse case notation for modeling vulnerabilities and insider attackers. The extended notation introduces a new type of actor called insider that has the capabilities and permissions of an authorized actor who misuses the given permissions. The concept of vulnerability is defined as a weakness that may be exploited by the misuse. In the extended modeling notation, the concept of vulnerability is expressed as a type of use case, with an exploit relationship from the threat to the vulnerability. However, use cases do not semantically express vulnerabilities as a weakness of the system Abuse case model is not semantically related to use case model; therefore, the abuse case model does not provide means to analyze the impact of an abuse case on other use cases. Actors and malicious actors could be the same entities or different, but the abuse case approach does not differentiate them.

Using the inverted strategic actors [51], one cannot explicitly model the impacts of countermeasures on attackers' goals and other goals, because the countermeasures are placed inside the boundary of the victim actor and the countermeasure analysis does not consider which player needs to employ the countermeasures.

Several conceptual modeling approaches such as [70, 79, 56, 65] employ tree style or graph-based constructs for modeling attacks. Graphs are useful for expressing the attack steps as nodes and events as edges. Trees are useful for decomposing the attack progress, causes, and events.

However, Attack Trees do not provide constructs to express required resources, access level, or skills required to perform an attack. Although Fault Trees enable modeling faults and tracing them to leaf events or errors, it does not provide means to express vulnerabilities of the system that lead to the faults and failure. Although minimum path-set are proposed for modeling ways to prevent occurrence of a failure, the impact of the countermeasures on other events, faults, vulnerabilities, attacks, and countermeasures cannot be expressed explicitly. The Attack Nets approach does not support countermeasure modeling and analysis. In addition, Attack Nets do not consider modeling and analyzing the concept of vulnerability and the relation between attacks and vulnerabilities. Although Attack Graphs are able to model the steps of an attack, its post and pre conditions, required configurations,

Table 1: Summary and comparison of security modeling notations. N indicates that the modeling notation does not consider the concept in its conceptual modeling constructs and Y indicates the concept is considered implicitly or explicitly in the notation.

Security Modeling Notations	Attack	Security Mechanisms	Vulnerability	Assets	Functional Requirements	Security Requirements	Other Quality Requirements	Usage
Attack Tree [70]	Y	N	N	N	N	N	N	Modeling and understanding attacks, identifying vulnerable points
Fault Tree [79]	Y	Y	N	N	N	N	N	Assess likelihood of the system failures
Attack Nets [56]	Y	N	N	N	N	N	N	Modeling steps, concurrency and attack progress
Attack Graph [65]	Y	N	Y	Y	N	N	N	Vulnerability and attack modeling and analysis in computer networks
Abuse Case [55]	Y	N	N	N	Y	N	N	Communicating the flaws and family of abuse cases of desired use cases to end users and customers.
Misuse case [75]	Y	Y	N	N	Y	Y	N	Modeling attacks in conjunction with functional requirements for security requirements elicitation
Extended Misuse Case [68]	Y	Y	Y	N	Y	Y	N	Modeling vulnerabilities to identify all possible threats and attacks
CORAS Risk Modeling Notation [10]	Y	Y	Y	Y	Y	Y	N	Model-based risk analysis
Secure Tropos [61]	Y	Y	N	Y	Y	Y	Y	Incorporating security concerns into the agent-oriented development process for security requirements engineering
Improvements on Secure Tropos [54]	Y	Y	Y	Y	Y	Y	Y	Security risk management in early phases of development
UMLsec [43]	N	Y	N	Y	Y	Y	N	Expressing security relevant information within UML diagrams
SecureUML [52]	N	Y	N	Y	Y	Y	N	Model-driven development of secure systems based on UML

and capabilities, they do not provide a way to express the impact of the attacks on system functionalities.

UMLsec goal is to define a universal set of stereotypes and tags that encapsulate security design knowledge to be used as part of UML diagrams. However, UML is not a requirements engineering notation, and the only diagram that focuses on the expected functionalities from the users point of view is the use case diagram. The formal semantic defined for use case model in UMLsec is limited to few stereotypes. The resulting models do not express attackers' behavior, and threat description is limited to using the notion of *Delete, Read, Insert* stereotypes to changes a state of the subsystem. Therefore, the usefulness of the modeling constructs is based on the expressiveness and comprehensiveness of the stereotypes. SecureUML can be used in the context of a model-driven software development process to generate access control infrastructures. However, the meta model of the SecureUML notation is limited to the access control concepts.

Using the approach proposed in [61], security related entities are tagged with an "s", while distinguishing the security entities from other systems entities does not affect the result of analysis on the models. The notion of threat introduced in the secure Tropos is limited to a visual representation, threats are not assigned to an attacker, and goals and refinements of the attacks are not expressed. In the enhancement of Secure Tropos notation in [54], threats are linked to the source actor and vulnerabilities are not related to the elements that bring them to the system. While impact of vulnerabilities on attacks is expressed, impact of countermeasures on vulnerabilities is not captured.

3 Security Requirements Frameworks

This section surveys the existing frameworks for eliciting, modeling, and analyzing security requirements. The goal of this survey is to investigate the main factors for driving security requirements. We study the role of conceptual modeling notations in elicitation and analysis of security requirements. We study how different approaches for deriving and expressing security requirements result in different expressions of requirements.

3.1 Agent and Goal-Oriented Requirements Frameworks

The notation of goal has been used in several security requirements frameworks. Lamsweerde [77] proposes a goal-oriented framework for generating and resolving obstacles to goal satisfaction. In this framework two models are developed iteratively and concurrently. First the model of the system-to-be, and secondly, the anti-model. Anti-model includes anti-goals, which are the attackers goals and malicious obstacles to security goals, set up by the attackers to threaten security goals. Anti-goals are refined to form a threat tree, in which the leaf nodes are either software vulnerabilities or anti-requirements. New security requirements are then obtained as countermeasures to the various vulnerabilities. Anti-goal method can be placed in the threat-based requirement engineering method groups as well.

Liu et al. [50] propose employing explicit modeling of relationships among strategic actors in order to elicit and analyze security requirements. In this approach, first generic role dependency patterns between actors in the domain are identified. This model can be elaborated to express whether the roles in the dependency relationship have trust, security, or privacy expectation from other. Then the roles that attackers can potentially play are considered and attacks to the system are modeled as negative contributions to dependency link. The agent-oriented analysis continues with elaborating the role-agent hierarchy and modeling the dependency derivation. This helps to find out what dependencies are inherited to the attacker if one of the roles plays as the attacker. In this approach, security requirements are originated from specifying security goals that actors expect from each other.

In a similar approach, Liu et al. [51] analyze attackers, vulnerabilities in actors dependency network, countermeasure, and access control. In this contribution, all actors are assumed potential attacker, which inherit capabilities, intentions, and social relationships of the corresponding legitimate actor. The basic idea of dependency analysis is that dependency relationships bring vulnerabilities to the system and the depending actor. The dependency vulnerability analysis aims to find which dependency relationship is vulnerable to the attacks. In this regard, the actors in the basic dependency model is substituted with its corresponding attacker, and then the impact of the attack to the dependency relationship is traced to the network of actors.

The contribution in [23] introduces the Secure Tropos[1], a formal framework for modeling and analyzing security requirements based on the concept of trust, ownership, permission, and delegation. These concepts are dealt within the normal functional requirements model. In this framework, ownership is defined as the relationship between an actor and a service if an agent is the legitimate owner of a service. Trust among two actors and a service means that the actor A trusts the actor B to fulfills a service S or achieve a goal G. Delegation among two actors and service means that the actor A explicitly delegates a goal, execution

[1]In security requirements literature, two different frameworks developed by different researchers are called Secure Tropos [61, 23].

of a task, or access to a resource to the actor B. The corresponding meta model of the notation is presented as part of the i* meta model in [82].

We described Securre Tropos modeling notation [61] in the previous section. In [59] security concerns are integrated into all phases of Tropos agen-oriented methodology: from early and late requirements, and architecture and detailed design. At the early requirements phase, Security Diagram is constructed and security constraints are imposed to the stakeholders. During the late requirements stage, security constraints are imposed to the system-to-be in the Security Diagram. The system is presented as one or more new actors, who have a number of dependencies with the other actors of the organization. In the architectural design stage, security constraints, secure entities that the new actors introduce, and secure capabilities are identified and assigned to each agent of the system. At the detailed design stage, agent capabilities and interactions are specified.

In a related work to Secure Tropos requirements framework, Bresciani et al. [12] suggest a quantitative security requirements analysis on the security constraints models developed using Secure Tropos method. The analysis considers measures of security criticality and complexity. Criticality is the measure of how the security of the system will be affected if the security constraint is not achieved. Security complexity is the measure of the effort required by the responsible actor for achieving a security constraint. The goal of analysis is to make sure that security bottlenecks are identified and an actor is not overloaded with security responsibilities. Figure 1 summarizes and compares the surveyed approaches and the main factors that each method employ to drive security requirements.

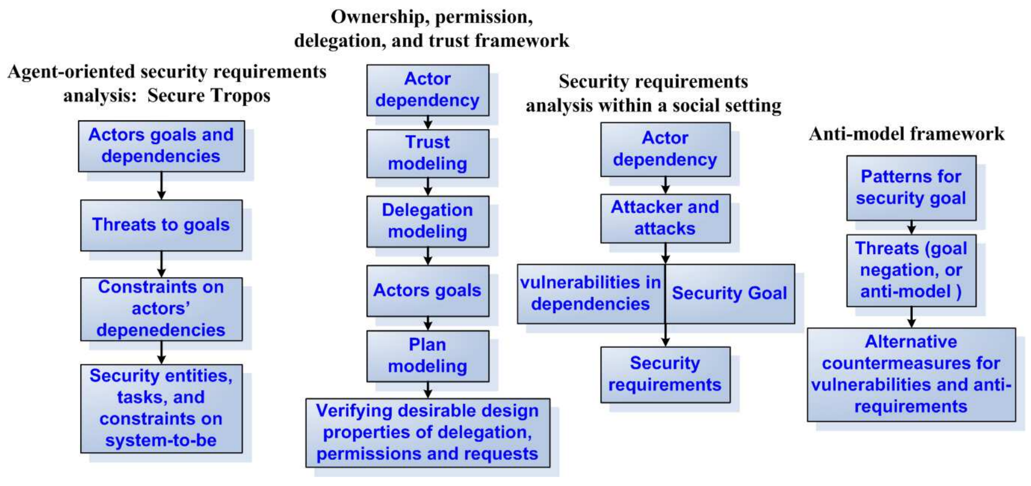
Agent-oriented security requirements analysis: Secure Tropos
Actors goals and dependencies
Threats to goals
Constraints on actors' depenedencies
Security entities, tasks, and constraints on system-to-be
Ownership, permission, delegation, and trust framework
Actor dependency
Trust modeling
Delegation modeling
Actors goals
Plan modeling
Verifying desirable design properties of delegation, permissions and requests
Security requirements analysis within a social setting
Actor dependency
Attacker and attacks
vulnerabilities in dependencies
Security Goal
Security requirements
Anti-model framework
Patterns for security goal
Threats (goal negation, or anti-model)
Alternative countermeasures for vulnerabilities and anti-requirements

Figure 1: Summary and comparison of agent and goal-oriented security requirements frameworks.

3.2 Trust-Based Requirements Frameworks

Viega et al.[80] argue that trust and trustworthiness are foundation of security, and the basis of trust relationships and trust formation can dramatically affect the underlying security of any system. They assert trust relationship between the entities must be formalized and mapped into the system requirements for implementation. Without recognizing all the entities and their trust relationships in a software system during the requirements phase of a project, that project is doomed from the start. They conclude that it is very important to minimize or eliminate the trust assumption between the various component in a multiparty system. Therefore, a recent shift in security requirements engineering is toward analyzing trust relationships and trust assumptions. Approaches such as [31, 28, 30, 29, 23] analyze effects of trust relationships and assumptions about trust on security requirements.

The contributions in [28, 30] focus on analyzing the trust assumption on deriving security requirements using problem frames [39]. The focal point of these contributions is the argument that trust assumptions can have fundamental impacts on how the system is perceived [28, 30]. In this regard, trust is defined as the quantified belief by a trustor with respect to the competence, honesty, security and dependability of a trustee within a specified context. By considering trust assumptions, a requirements engineer believes and accepts that the domain holds some certain properties and specification. To incorporate the trust assumption into the problem frames models, which is used as the requirements elaboration modeling technique, the problem frames notation is extended with an arc from domain to a dotted oval describing the properties assumed to be true [30]. By adding trust assumption the model works as a documentation about the way that requirements engineer trusts the behavior of the domain, and the scope of the analysis is limited to the domain.

Considering trus assumptions, Haley et al. [29] propose a security requirement engineering framework for expressing and analyzing security requirements as constraints on functional requirements. The framework introduces four main activities to move from functional goals to final arguments that if the security requirements are satisfied. The activities start with identifying functional requirements. In the second step, security goals and security requirements are identified. Finally, satisfaction arguments are constructed. A set of security requirements core artifacts [42] is introduced which are developed during the discussed activities and mainly consists of documenting application and security goals, assets, management control principles, functional, quality, and security requirements, and system architecture. Security goals are operationalized into security requirements, aiming to protect the assets from harm. Security requirements are treated as constraints on the systems functional requirements. Figure 2 summarizes these group of works based on the trust assumptions.

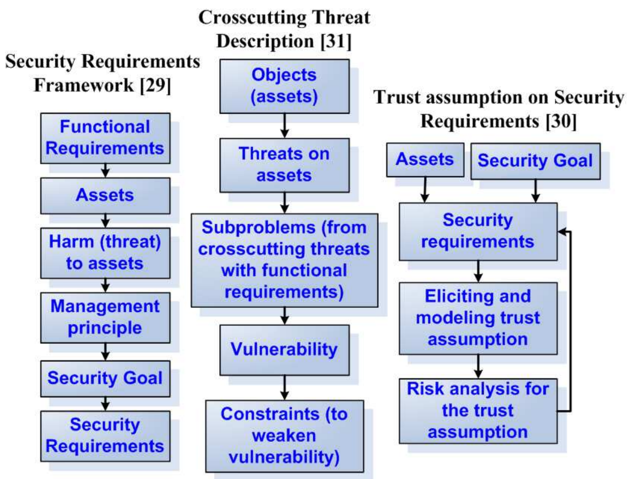
Security Requirements Framework [29]
Functional Requirements
Assets
Harm (threat) to assets
Management principle
Security Goal
Security Requirements
Crosscutting Threat Description [31]
Objects (assets)
Threats on assets
Subproblems (from crosscutting threats with functional requirements)
Vulnerability
Constraints (to weaken vulnerability)
Trust assumption on Security Requirements [30]
Assets
Security Goal
Security requirements
Eliciting and modeling trust assumption
Risk analysis for the trust assumption

Figure 2: Summary and comparison of threat and trust-based security requirements engineering methods in [31, 29, 30].

The apporach in [23] can be categorized also as a trust-based requirements engineering method. However, this approach mainly focuses on eliciting access control requirements based analyzing ownership, permission, and delegation relationships in a trust chain of actors.

3.3 Risk and Threat-Based Requirements Frameworks

Several security requirements engineering methods focus on analyzing threats and unwanted incidents for deriving security requirements. For example, Lamsweerde [77] suggests obtaining security requirements as countermeasures for anti-requirements and vulnerabilities which are output of analysis of anti-goals and threat trees.

In another work by Haley et al. [31], threats are used as crosscutting concerns to determine the impact of security requirements on the functional requirements. Threat descriptions are crosscut with the subproblem to determine which threats apply to a given subproblem. Threats are composed with subproblems looking for vulnerabilities that may be used to exploit a threat. In this approach, security requirements are expressed as constraints on functionalities or trust assumptions.

Evans et al. [18] suggest Mission Oriented Risk and Design Analysis (Morda) as a methodology for analyzing security risks. Morda combines threats, attacks, and mission impact concepts for deriving an unbiased risk metric. The analysis starts with defining the system and threats. Then, relevant missions and impacts of adversary on them are identified. To analyze the adversary, the objectives of attacks are identified and an attack tree is developed for the attack. Morda suggests calculating attack score to assess the risks. The combination of attack scores represents the adversarys preference for a portfolio of attacks against the systems missions. This parameterized attack portfolio defines the overall system risk in terms of calculated utility scores. Finally, security engineers derive countermeasure alternatives first by focusing on the highest-scoring attacks and characterizing potential countermeasures according to their costs and benefits. In [19], Evans and Wallner argue that attack score provides the metric necessary for making trade-off decisions. In this method, design decisions are made by comparing the changes in risk, cost, and performance parameters.

Risk management methods are frequently used as part of the secure software development activities. Several methods such as Security Attributes Evaluations Method (SAEM)[13], CORAS UML profile and methodology [34, 15, 10], extensions to Tropos for risk modeling [4], improvements on Secure Tropos for risk assessment by Matulevicius et al. [54], and the risk-based security requirements engineering framework [58] offer modeling and analysis approaches for security risks assessment.

Among these methods, the framework proposed by Mayer et al. [58] focuses on integrating risk analysis with requirements engineering activities. This risk-based security requirements engineering framework is concerned about integrating requirements engineering practices with security engineering, as well as intertwining between requirements and architecture design. The main idea is to align IT security with business goals. For this aim, the impacts of risks on business assets are analyzed, risks are related to the threats and vulnerabilities in the architecture, and security requirements are identified to mitigate the risks.

In this approach, risk is defined as the combination of the probability of occurrence of harm and the severity of that harm. IT risk are further decomposed into three components: $Risk = Threat * Vulnerability * Impact$. In this way, risk is characterized by the opportunity of exploiting one or multiple vulnerabilities, from one or many entities, by a threatening element using an attack, causing an impact on business assets. The proposed approach employs the i* Framework as the requirements and architecture modeling method, and adapts risk analysis in part of the early security engineering practices. New extensions to the i* are introduced to express threats, vulnerabilities that the threat exploit, and the relation of the vulnerability with elements of the model.

We described CORAS modeling language in the previous section. In addition to the UML-inspired modeling notation, CORAS provides a methodology based on the unified process for conducting security analysis [15]. The CORAS risk assessment methods is adapted to address requirement elicitation. In [10], a seven-steps risk analysis method based on the CORAS modeling approach is presented. The analysis method consists of analyzing the target context by developing UML models and asset model (refer to the previous section for details of the modeling notation). Then, potential players who impose the risks, target assets, and vulnerabilities that make the risks possible are identified. The risks that have sever consequences and high likelihood are selected for further analysis. For detailed risk analysis, threat scenario diagrams are developed and unwanted incidents are identified. Finally, treatments are selected for the risks that not acceptable. In section 4, other risk assessment methods that do not focus on security requirements engineering are surveyed in detail.

3.4 UML-Based Requirements Engineering

UML is a widely used notation in analysis and design of software systems, developers already know UML modeling notation, and the notation is well defined. Therefore, UML and extensions to the UML has been used for security requirements engineering in methods such as [75, 55, 43].

As described in the previous section, UMLsec is an extension to UML that allows expressing security relevant information within UML diagrams. Using UMLsec, security requirements are defined by assigning security stereotypes to the elements of the design models. Since UML is not a requirements engineering notation and the only diagram that focuses on the expected functionalities of the system from the users' point of view is the use case diagram, several security requirements engineering methods based on use cases are proposed.

Jurjens [44] combines the use of UMLsec modeling, use-case driven process, and goal trees to design the system along with modeling functional and non-functional requirements respectfully. In this method, the goal tree is developed to record the result or reasons of design actions which are expressed in UMLsec diagrams. The security goals are refined in parallel by giving more system details, such as UMLsec stereotypes or tag-values, in design phases.

We described the misuse case and abuse case modeling notations in the previous section. The process of eliciting security requirements by misuse cases [75] starts with identifying critical assets. Then security requirements for each asset are defined. In the third step, threats to each security requirements are defined and expressed as misuse cases. In the fourth step, risks of threats are identified and analyzed. Finally, security requirements are defined as either security use cases or in the mitigation field of misuse case description. Sindre and Opdahl [75] assert that the visualization of links between use cases and misuse cases help to organize the requirements specification and tracing requirements to threats that motivated them. The authors also propose templates for detailing misuse cases to support requirements extraction [74].

Firesmith [20] also employs the notion of use cases and misuse cases for analyzing security requirements. He claims use cases are typically misused to unnecessarily specify security architectural mechanisms instead the security requirements. Therefore, he suggests a template for describing reusable security use cases in [20] for specifying security requirements.

Another example of UML profiles for security analysis is the CORAS framework. CORAS, inspired by UML diagrams, employs similar models to use case models for expressing assets, risks, vulnerabilities and security treatment. General UML diagrams are also used in the CORAS methodology for modeling the target system context [10].

3.5 Discussion

The final outcome of security requirements engineering frameworks is expected to be a list of feasible requirements that provide good-enough system's security against potential attacks. To extract the list of feasible security requirements requirements analysts need to consider possible alternative solutions that satisfy the requirements. Since the output of security requirement engineering is a set of required protection mechanisms and constraints on the system-to-be, the need to parallel elaboration of the requirements and architecture design is pointed in various works [31, 66, 63]. Lamsweerde [77] points to other meta-requirements for a security requirements engineering technique such as: suitable for early development phases and high assurance, incremental, enabler for reasoning about alternatives and security-by-construction, and separating security concerns for interaction analysis.

Existing security requirement engineering frameworks and modeling notations mostly focus on identifying security requirements and evaluating if a certain system design can satisfy those requirements. For example, UMLsec [43] focus on encoding requirements and design in system level details. Although employing different modeling notations, existing security requirements engineering frameworks consider similar concepts such as assets and threats to drive security requirements. Surveying several security requirements engineering frameworks shows that since existing approach take different views to model systems and security, they result indifferent types of security requirements. For example, some approaches define system as a network of strategic and social actors that delegate services to the other actors and may (not) trust each other [51, 23, 61]. In approaches such as [43, 29, 31, 30], the system-to-be and the environment are modeled by focusing on the different aspects of the system from one single point of view. Using UMLsec [43], one can conclude if a system setting satisfies a set of requirements expressed as stereotypes, while analyzing actors dependencies from point of view of trust, permission, ownership, and delegation ownership [**?**] results in conclusion about dependencies among partners and access control requirements.

Security requirements engineering frameworks do not agree on the concept of security. While UMLsec [43] enables modeling and analyzing issues such as *fair exchange* and *secrecy* as the security concerns, approaches such as [23, 51, 61] deals with security as an issue that arises from the interaction of social actors; approaches such as misuse cases [75] and abuse cases [55] deal with security as an issue where the system users misuse the functionality; framework in [29] considers security as a constraint on the functionality; and risk-based requirements frameworks [10, 58] extract security requirements to mitigate the risks.

In sum, the output of existing security frameworks include a variety of concerns such as: which trust assumptions are invalid, what kind of constraints need to be enforced on functionalities, what kind of attacks need to be prevented, what type of security countermeasures need to be employed, and what vulnerabilities need to be patched. Table 2 summarizes and compares the similarities and differences of exitisng security requirements engineering frameworks:

4 Security Trade-off Analysis

Security, like beauty, is in the eye of the beholder.

Devanbu and Stubblebine [16]

While software systems act as enablers, security is a feature of the software that prevents certain actions. Security requirements are additional requirements which root from the risk of adversaries and vulnerabilities rather than stakeholders' interests. Therefore, security always involves a trade-off, which requires trading convenience, performance, privacy, etc. for better security. Trade-off analysis is a systematic examination of the advantages and disadvantages of each proposed requirements and/or design approach for a system to achieve the right balance among several competing goals [2].

A topic of increasing importance is the analysis and management of dependencies among requirements which is called Requirements Interaction Management (RIM) [67]. The satisfaction of one requirements or the environment can aid or detract from the satisfaction of another requirements [78]. When some goals are not sufficiently satisfied, designers need to explore further alternatives that can better achieve those goals without detrimentally hurting others. In addition, while a design solution satisfies some quality goals, it may have negative impacts on satisfaction of other requirements. This situation causes trade-offs between the positively- and negatively-affected requirements. Satisfying security requirements involves trading design objectives or values, such as performance, usability, functionality, and money for security.

Security trade-off are naturally subjective, since security decisions are based on personal judgment [71]. Different people have different level of risk tolerance and personal privacy expectations. In addition, different stakeholders have different expectation from a software system; therefore, they impose different security requirements which may conflict with security and other goals of other stakeholders. Analyzing costs and benefits of security-critical

Table 2: Summary and comparison of security requirements engineering frameworks. N indicates that the modeling notation does not consider the concept or analysis in its conceptual modeling constructs, and Y indicates the concept or analysis is considered implicitly or explicitly in the framework.

Security Requirements Framework	Asset	Attack Analysis	Security Social Concerns	Trust Concerns	Vulnerability Analysis	Security Constraints	Security Goals	Alternative Countermeasures Analysis	Risk Analysis
Misuse Case Analysis [75]	Y	Y	N	N	N	N	Y	N	Y
Abuse Case Analysis [55]	N	Y	Y	N	N	N	N	N	N
CORAS Framework [10]	Y	Y	N	N	Y	N	N	N	Y
Anti-Model Analysis [77]	N	Y	N	N	Y	N	Y	Y	N
UMLsec Framework [43]	N	N	N	N	N	N	N	N	N
Social Actor Analysis [51]	N	Y	Y	N	N	N	Y	N	N
Trust, Ownership and Delegation Analysis [23]	N	N	Y	Y	N	N	N	N	N
Secure Tropos [61]	N	Y	Y	N	N	Y	Y	Y	N
Security Requirements Engineering by Haley et al. [29]	Y	Y	N	Y	N	Y	Y	N	N
Crosscut Threat Description [31]	Y	Y	N	Y	Y	Y	N	N	Y
Trust Assumption Analysis [30]	Y	Y	N	Y	N	Y	Y	N	N
Risk-Based Security Framework by Mayer et al. [58]	N	Y	N	N	Y	N	Y	Y	Y

systems may also involve ethical issues. In a context that security attacks may pose risks to human life, safety, and privacy, quantification of human loss or privacy violation is not ethically acceptable [9].

Although there is an agreement that security is about trade-offs [71, 69], few software engineering research contributions have considered trade-off analysis as a requirements engineering and design activity. While risk assessment methods such as [34, 13] address balancing the costs and benefits of security solutions to achieve good enough security, trade-offs that security goals and alternative security solutions impose on other quality objectives are neglected in existing security requirements engineering frameworks. In this part, we discuss the psychology aspects of security trade-offs to draw conclusions on the requirements for a security trade-off analysis method. Then a survey and comparison of trade-off and risk analysis methods is provided.

4.1 Psychology of Security and Security Trade-offs

Bruce Schneier [72] uses the term *psychology of security* to elaborate the way that humans decide on security risk and make trade-offs. He asserts that security is a feeling based not on probabilities and mathematical calculations, but on the psychological reactions to both risks and countermeasures. Three fields of research: behavioral economics, psychology of risk, and neuroscience are considered related for studying the human perception of the risks. Behavioral economics looks at the human emotional, social and cognitive biases and how that affects economic decisions. Studies on sychology of risk try to find out when humans exaggerate risks and when they underestimate them. Neuroscience is important to understand how we think about security risks.

Schneier concludes that the approaches for designing security systems and deciding on security trade-offs need to take the feeling and psychology of security into account. Humans do not evaluate security trade-offs mathematically, by examining the relative probabilities of different events. Instead, they use shortcuts, stereotypes, biases, generally known as heuristics. Empirical studies on trade-off decisions show that unlike the utility theory, people have subjective values for gain and losses, which is described by *prospect theory* [45]. In addition, peoples' choices are affected by how a trade-off is framed. By framing a choice as gain, people tend to be risk-averse; in opposite, when the choice is framed as loss, people tend to be risk-seeking. This is called *framing effect*.

Other biases also affect risk analysis. One common risk bias is called *optimism bias*: people tend to believe they will do better than most others engaged in the same activity and bad things do not happen to them. People also are more likely to accept risks if they feel they have control over them; this bias is called *control bias*. The *affect heuristic* also says that an overall good feeling toward a situation leads to lower risk perception.

Human minds deal with heuristics and probabilities that may lead to wrong trade-offs. For example, in human mind, small numbers matter much more than large numbers. While people are good with comparisons such as 1 vs. 2, 4 vs. 8, they are less good at decision about 10,000 vs. 100,000. The other heuristic is called, *availability heuristic* which states that in any decision making process, easily remembered and available data are given greater weight than hard-to-remember data. The other heuristic involved in decision making is *Cost Heuristics*. People use a mental accounting which assigns different values for different charges that have the same cost in terms of money. Schneier states that the effect of mental accounting on security trade-offs is not clear but people have mental accounts for safety and security, and the money spent from that accounts feel different than money spent from another account.

The result of trade-off decision making depends on other heuristics such as context (*context effect*), variety of choices (*choice Bracketing*), random information provided to the decision maker (*anchoring effect*), etc. While some of these heuristics and biases may lead to wrong trade-offs, considering them in developing a trade-off analysis method is important. The example of Utility Theory vs. Prospect Theory shows that making proper trade-offs do not rely only on a mathematically sound formula, but also on the human's mind preferences. Although the possibility of gain and loss might be equal, humans prefer certain choices. The question is to what extend such human mind preferences should be incorporated into the decision making process and to what extend such biases need to be avoided in the trade-off analysis processes?

4.2 Security Trade-off Analysis: State of the Art and Practice

This section first provides an overview of the trade-off analysis and risk assessment methods. then, se survey possible ways to express trade-offs between design objectives and trade-offs of alternative solutions. This section discusses why trade-off analysis among quality objectives is challenging and what conceptual constructs are involved in modeling and analyzing security trade-offs.

4.2.1 Cost-Benefit Analysis and Risk Assessment Methods for Trade-off Analysis

Cost-benefit and risk analysis have been considered as part of software engineering activities and specially important in security requirements engineering and secure software design. In security engineering, risk involves probability of the threat and seriousness of a successful attack [37]. There exist various methods for modeling and incorporating risk assessment along with cost-benefic analysis into the software engineering practices.

The risk-based security requirements engineering framework proposed in [58] is concerned with integrating requirements engineering practices with security engineering and interleaving requirements and architecture design by adopting risk analysis practices as a key activity in the early stages of development. The approach aims to analyze the impacts of threats on the business by linking the risk impacts with the business values. This approach uses and extends the i* Framework for modeling the business context, system architecture, threats, and vulnerabilities. The i* modeling notation is extended to express the threats that target actors' goals by using vulnerabilities. Risk analysis involves considering business assets, vulnerabilities, probability of attacks, and cost of countermeasures. The main criteria to make trade-offs and sort the indicators are risk values and cost values.

The CORAS framework [34] provides a UML profile for risk assessment. The proposed profile defines UML stereotypes and rules for specialized UML diagrams to support of model-based risk assessment. Asset diagram is introduced to identify assets; threat diagram captures unwanted incidents causing loss in asset value; state analysis diagram is proposed to estimate and document the frequency and consequence of the unwanted incidents; and treatment diagrams are models extended with specialized use cases representing treatments.

Asnar et al. [4] introduces a method for modeling and analyzing risks at organizational level. The proposed framework is called Goal-Risk Model and extends the Tropos methodology with three basic layers: strategy (or goal layer), event, and treatment. Strategic layer analyzes strategic interests of the stakeholders. Event layer analyzes uncertain events along their impacts to the strategy layer, and treatment layer analyzes treatments to be adopted in order to mitigate risks. This framework is extended in [5] to assess the risk in a multi actors setting. In the extended framework, the concept of actor and dependency between actors are considered besides the concepts of goal, task, and event.

The Goal-Risk Model provides the basis to analyze if the requirements are satisfied and the risk level is acceptable for every actor. For this aim, first goals are operationalized to analyze actors' goals and the tasks used to achieve them. Then, events are operationalized to analyze events and their impact on the strategy layer. This permits to do trade-off analysis when an event acts as a risk for some goals and as an opportunity for other ones. Then, the risk level perceived by each actor in the organization is calculated. Finally, treatment Operationalization intends to refine the Goal-Risk Model in case the risk-level is higher than the risk acceptance defined by actors. Risk assessment involves considering factors such as evidence values for satisfaction or denial of tasks and goals; the costs events and tasks; utility values of higher goals; the likelihood of local and global events; and the risk tolerance for each actor.

Another approach for deciding on security requirements and design is proposed in [41], where probabilistic inference on security influence diagrams and Bayesian Networks are used to support trade-off analysis. In this work, authors discuss the criteria of the language for security requirements modeling and analyzing. They assert that the language should be able to represent decision's maker goals, domain of control, and causal relation between goals (indirect and controlled). For managing the definitional uncertainty the language must specify what is meant by the conceptual concepts of the language, and must represent the causal and empirical uncertainty.

Butler [13] suggests a cost-benefit analysis method called SAEM to compare alternative security designs. SAEM relies on a quantitative risk and benefit assessment in which the analysts need to interview IT and security managers to elicit initial data for the analysis. The analysis starts with the assessment of security technologies benefits. Benefit of a security technology is an assessment of how well the technology mitigates a risk. The mitigation may work in two ways: prevent the attack from occurring or reducing the consequences of an attack. The usage of security technologies are divided into three main categories: protection, detection, and recovery. Then, to assess the benefits of the technologies, threats that the technologies mitigate are identified. At this stage, interviews are required to capture a rough estimation about the quality and effectiveness of countermeasures. The results of technologies effectiveness are percentage indicating how a threat is reduced by a security technology.

In the second step of SAEM, the benefit assessment is applied to the threat frequencies and outcomes to determine how the overall threat index is affected. Threat index indicates the frequency of an attack and its probable outcomes. Consequences of an attack are called attribute, and the outcome is a vector of attributes where the value of the attribute is the level of damage. Since each security technology reduces the risk from several threats, comparing technologies requires an overall assessment of the threat index. Hence, new threat indices for alternative security technologies are calculated. In addition to considering the total changes of threat index, in the third step, alternative technologies are compared based on the coverage of prevention, detection, recover, and coverage of several threats. Finally, in the forth step, cost of alternative security technologies is considered for trade-off analysis.

4.2.2 Software Quality Trade-off Analysis Methods

There exists trade-off analysis methods that go beyond cost-benefit analysis and consider goals of stakeholders and quality attributes of the system. Kazman et al. [7] introduce a framework for modeling quality attributes and architectural alternative designs using the notion of scenarios and tactics respectively. A quality attribute scenario is a quality-attribute-specific requirement, and consists of six parts: Source of stimulus, Stimulus, Environment, Artifact, Response, and Response measure. Achievement of quality scenarios relies on tactics.

ATAM and CBAM. Architecture Tradeoff Analysis Method (ATAM) [46] is a labor-intensive evaluation method to analyze whether an architecture decision satisfies particular quality goals. ATAM helps designers to prioritize scenarios and evaluate alternative tactics using a Quality Attribute Utility Tree. Scenarios that have at least one high priority of importance or difficulty are chosen for a detail analysis to examine if the selected tactics satisfy the scenario.

The result of the ATAM analysis is an Architectural Approach Analysis table for each quality scenario. Evaluators identify and record sensitivity, tradeoff, risks and non-risks points of alternative tactics for satisfying a scenario in this table. Sensitivity and tradeoff points are architectural decisions that have effects on one or more quality attributes, the former positively and the latter negatively. In ATAM, a risk is defined as an architectural decision that may lead to undesirable consequences. Similarly, a nonrisk is an architectural decision that upon analysis is considered safe.

Cost Benefit Analysis Method (CBAM) [7] picks up where ATAM leaves off. It takes ATAM outputs as input and adds cost factors to the trade-off analysis. CBAM uses scenarios as a way to concretely express specific quality attributes. Stakeholders vote for scenarios to select the top priority scenarios for further analysis. In this way, CBAM resolves the conflict of goals among stakeholders by computing the ROI of each strategy based on their votes for utility of each strategy. Stakeholders' vote is used to calculate stimulus-response utility per each scenario for creating the utility-response curves. Utility is based on the importance of each scenario and its response value. The overall utility is combined with the cost of an architectural strategy to calculate the final Return On Investment (ROI) measure. The ROI value for each architectural strategy is the ratio of the total benefit to the cost. Using the ROI value, the architectural strategies can be rank-ordered and compared, and a final decision is made.

Using ATAM, in "Architectural Approach Analysis" table for each scenario, alternative tactics are analyzed. However, the impacts of tactics are analyzed on one specific scenario rather across multiple scenarios. Moreover, ATAM does not incorporate the degree of sensitivity or trade-off between tactics and qualities in the analysis, since the Architectural Approach Analysis table does not express the impact of tactics on stimuli (attacks) and does not specify the consequences of applying a tactic on other quality scenarios. The ATAM process does not separately capture the impact of each tactic on stimuli of security scenarios (attacks).

ATAM does not consider cost as a trade-off factor that usually has the highest priority over the other factors. CBAM is proposed to address this weakness. The CBAM approach to trade-off analysis requires the designer to obtain quantitative measures of current response and expected response of architectural strategies for each affected scenarios. The major obstacle toward applying CBAM would be unavailability and inaccuracy of such measures in the early stages of development and lack of agreement on metrics and measurement methods.

SVDT and AORDD. Security Verification and security solution Design Trade-off (SVDT) [36] framework provides methods for evaluating alternative security solutions and balancing security requirements. In this framework, security requirements and candidate security solutions are modeled and verified using UMLsec. The trade-off analysis is implemented using Bayesian Belief Nets (BBN) and makes use of a fixed set of trade-off parameters, such as budget, business goals, and time-to-market constraints, to support design decisions regarding how to best meet security requirements while also fulfilling other diverse system requirements.

SVDT consists of four main steps: (1) identify potential misuses and assess their risks, (2) identify alternative security solutions for misuses that must be addressed, (3) perform UMLsec security verification of these security solutions to ensure they prevent the target misuse and (4) perform security solution design trade-off analysis among the alternative solutions.

The first two steps are performed using the CORAS risk assessment platform [34]. CORAS is a risk assessment model-based methodology which provides a UML profile for risk assessment. In the third step of SVDT, the alternative security solutions for a particular misuse are modeled and verified using UMLsec. In this way, complete analysis is reserved for the most promising alternatives. In the fourth step, the trade-off analysis method incorporates the notion of a static security level to address the security level derived from development activities, and a dynamic security level to address the security level derived from predicting events when the system is operating.

The static security level, dynamic security level, standards, policies, laws and regulations, priorities, business goals, security risk acceptance criteria, time-to-market, and budget constraints are all parameters to the trade-off analysis, which computes the Return on Security Investment (RoSI) for each security solution. Security requirement, solution effect, solution cost, misuse cost and misuse impact are input parameters for RoSI computation as the information that is traded off. The goal of computing RoSI is to measure and compare the level of security of alternative solutions.

SVDT framework introduces the structure of trade-off analysis in form of a Bayesian Networks topology. The trade-off parameters are estimated and fed into the BBN to computer RoSI for each alternative solution. The result of risk assessment in this framework is a set of misuses and alternative security treatment strategies. The input to trade-off analysis is the results from risk assessment. SVDT framework is based on the Aspect-Oriented Risk-Driven Development (AORDD) [35] cost-benefit risk analysis approach for making security trade-offs. The AORDD also takes advantage of BBN for computing RoSI. The AORDD framework provides an ontology for security risks concepts and the BBN topology covers those concepts. In this framework, UML models of the system are annotated with stereotypes which help the analysts to select the right estimation set from the estimation repository. The annotations are then mapped to the BBN nodes and subnodes.

The use of probabilistic reasoning by BBN in SVDT (and AORDD) requires the software designers obtain quantitative measures or qualitative scales of the impact of misuses and solutions. The major limitation is the inaccuracy or unavailability of quantitative data on the impact of misuses and solutions especially in the early stages of the development lifecycle. In addition, requirements such as security, privacy, and usability are personal and subjective qualities. For example, different personalities have different level of risk tolerance and different people have different privacy expectations. Therefore, calculating one single value for security benefits or privacy costs of a solution may not be feasible in all circumstances.

A before head-designed BBN that returns the RoSI in every context saves time and efforts for many projects, while shifting the challenge to developers of such BBN that is applicable in every context. relying only on the RoSI metric to compare two alternative security design solutions, when the system designers and analysts need to trace the impact of changes in BBN parameters or changes in the design, they have to learn about the topology of the BBN which they have not designed.

Use of Misuse Cases for Trade-off Analysis. Another approach that aims to express trade-offs between system's objectives more explicitly employs the notion of use cases and misuse cases [2]. This approach visualizes the structure of the trade-off situation accurately and in a way that emphasizes the essential points of conflicts and trade-offs. For this aim, two new types of relationships to the misuse case model are added: 1) Mis(use) case A *aggravates* misuse case B, which means A increases either the probability of success or seriousness of damage that B threatens. 2) Use case A *conflicts with* use case B, if achieving A's goal makes achieving B's goal more difficult or impossible.

The relationships in (mis)use case model, *threatens*, *mitigates*, *aggravates*, *conflicts with*, together with includes and has expectation (extends), provide the toolkit to handle trade-offs. This approach enables the analyst to describe threats, and discover functional and non-functional requirements. By visualizing the impact of misuse cases and conflicting use cases with easy-to-understand relationships, the model show areas where costs and benefits can be expected. The main limitation of this approach is the lack of structures that relate the causes of trade-offs to use cases, since a use case model is limited to describe the expected functionality rather than the architectural solution. Therefore, the misuse case model does not express due to what operationalizations the use cases are in conflict.

Schneier 5-Steps Trade-off Analysis Method. Bruce Schneier suggests a five-steps trade-off analysis method in [71], which helps security engineers to assess threats, risks, security solution, and trade-offs imposed by a security solution. In the first step, the analyst studies the asset that security solutions are trying to protect. In the second step of the trade-off analysis, the analyst needs to ask what are the risks to the assets? In this step, the analyst considers who wants to attack the system, how they might attack it, and what are the consequences of the successful attacks. In the third step, the security solution is evaluated in terms of its impact on the risks and how well it mitigates those risks. In the next step, other risks that the security solution causes are studied. Finally, in the fifth step, the costs and trade-offs that the security solution imposes are studied. This means considering the downsides of applying the security solution, such as costs in terms of money, matters of convenience and comfort or privacy.

4.3 Discussion

The surveyed trade-off modeling and analysis approaches express trade-offs at different levels of explicitness and abstraction. In many engineering fields, mathematical formulas model the relationships between trade-off parameters. Using the AORDD and SVDT methods, trade-off parameters are expressed in a BBN where the BBN topology visually expresses the relation between trade-off parameters. In this way, a single fixed formulation of relationships between trade-off parameters is reused for multiple projects. Using the ATAM and CBAM approaches, trade-offs are recorded by the lists of tactics, scenarios and quality attributes in tables. Trade-offs are analyzed by an architecture expert that evaluate each design tactic in terms of its costs and benefits. Using the misuse case models for trade-off analysis, one can explicitly express the (mis)use cases that have negative or positive impact on other (mis)uses and use cases that have conflicts. Although trade-offs are expressed explicitly, the (mis)use case model is not expressive enough to model the trade-offs causes.

Figure **??** depicts the different levels of abstraction for expressing trade-offs and their properties. Table 3 provides a summary and comparison of the existing trade-offs modeling and analysis approaches.

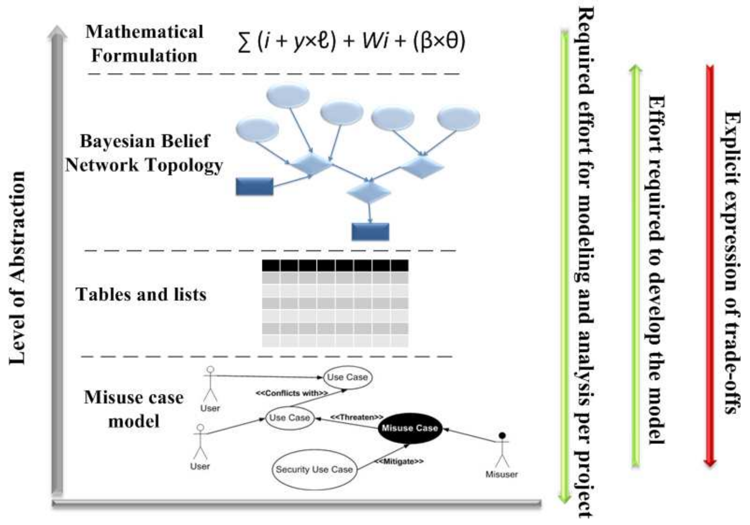

Figure 3: Abstraction level of different modeling approaches for expressing the relation between trade-off factors.

5 Security Knowledge Management

Many recurring patterns of attacks and vulnerabilities have been identified by longtime software security practitioners [57]. In addition to awareness about potential attacks, designing security-critical systems requires knowledge and security expertise in various fields such as computer networks, operating systems, communication protocols, cryptography algorithms, and access control methods. Barnum and McGraw [6] note that "software security does not have enough master craftsmen to effectively apprentice and train the worlds developers, architects, and software security newbies" and software designers usually lack such expertise and skills, and security knowledge is hard to acquire.

Mead and McGraw [57] assert that one of the challenges facing software security is the lack of an easily accessible common body of security knowledge. Although much security knowledge is widely available in the form of textbooks, checklists, standards, and security design patterns, it remains difficult for designers to extract relevant pieces of knowledge to apply to their specific design or requirements related decision making situations. Structuring knowledge helps the knowledge consumer browse the content and finds the relevant information more efficiently.

There are various approaches for collecting, structuring, and managing security knowledge such as taxonomies, ontologies, standards, guidelines, checklists, patterns, repositories, and web-based data bases. This section provides an overview of advantages and limitations of various approaches for structuring and sharing security-related knowledge. We divided the knowledge management contributions for security into two main groups: 1) A group of work that suggests a way to structure and organize knowledge into a specific format, 2) A group of work that provides the body knowledge in form of a taxonomy, categorization, repository, or data base. The first group of contributions is concerned about defining the knowledge structure. The second groups of work, which are more common, may introduce a way to structure knowledge but also provides the actual populated knowledge in a structure.

5.1 First Group: Knowledge Structures

Barnum and McGraw [6] have suggested schema of a security Knowledge Base (KB), in which they identify seven knowledge catalogues: principles, guidelines, rules, attack patterns, historical risks, vulnerabilities, and exploits, which are categorized into three categories: prescriptive, diagnostic, and historical. Prescriptive knowledge category, including

Table 3: Summary and comparison of trade-off modeling and analysis methods.

Method Properties	ATAM/CBAM	SAEM	Misuse Case	SVDT/AORDD
Model developer	Modeled by the project analyst	Modeled by the project analyst	Modeled by the project analyst	Modeled by a project-independent analyst beforehead
Level of the abstraction of the model	Trade-offs are expressed by relationships in a table	Trade-offs are modeled and analyzed by relationships in tables	Trade-offs are expressed by relationship between the (mis)use cases	Trade-offs are formulated in the BBN topology
Modeling Constructs	ATAM/CBAM	SAEM	Misuse Case	SVDT/AORDD
Design objectives and goals	Expressed in terms of scenarios	Limited to security benefits and costs (expressed in tables)	Expressed as use cases	Considered by fixed BBN parameters
Relations between objectives	Not considered explicitly	The relation between cost and security objectives are expressed by quantitative measures in tables	Expressed by *include*, *extend*, $conflictswith$	Considered in the fixed BBN topology
Alternative design solutions	Expressed in terms of tactics	Listed in tables	Design solutions are not modeled	Each alternative security solution is analyzed by the UMLsec reasoning and BBN cost benefit analysis
Impacts of objectives and solutions on each other	Utility tree in ATAM doesnt capture the contributions of scenarios on each other/ CBAM considers the impacts by utility and side effects of an architectural strategy	Limited to impact of solutions on risks and costs by effectiveness estimates	The impact of (mis)use cases are expressed by *aggravate*, *mitigates*, and *threaten* relations	Encoded in the BBN topology
Extents of objectives achievement	Not considered in ATAM/ Expressed in terms of Utility-Response Curves in CBAM	The extent to which threats are controlled are expressed by threat indices	Not modeled	Expressed by parameters fed into the BBN and mostly quantitatively
Inaccurate or incomplete knowledge	Not considered	The extent to which threats are controlled are expressed by threat indices	Not considered	In terms of probability density functions (pdf) in BBN
Multiple actors	Expressed implicitly by multiple stimuli sources in ATAM. In CBAM multiple stakeholders votes are taken into account	Not considered	Multiple actors' use cases are expressed from view points of several actors	The BBN topology can contain parameters from multiple actors point of interest
Security specific forces	Not considered	Threats and their risks are considered	Misuse cases and the relations of *threatens* models the attacks and threats	Some concepts are considered in the BBN topology

principles, guidelines, and rules offers advice for what to do and what to avoid. Diagnostic knowledge, including attack patterns, exploits, and vulnerabilities help the designers identify and deal with common problems that lead to security attacks. Historical knowledge includes catalogue of historical risks.

NFR security catalogues [14] is another example of security knowledge catalogue. The NFR framework is a framework for modeling and analyzing non-functional requirements, using the notion of softgoals. Softgoals are goals that have no clear definition or criteria of satisfaction, like security. The interaction among security requirements and other non-functional requirements are modeled by correlation rules. The NFR framework introduces operationalization methods, which are ways to achieve security requirements. Catalogues of correlation rules facilitate the systematic capture and reuse of knowledge of softgoal harmonies and conflicts and trade-offs among operationalization methods.

Gross and Yu [26] present an approach to reorganize knowledge in a design pattern according the contribution of alternate solutions to various design goals. The design goals are typically non-functional requirements (NFRs) such as performance, maintainability, and security, which may conflict with each other; thus requiring trade-offs. To characterize patterns according to the NFRs, this approach suggests modeling the impact of each design pattern to other NFRs to explicitly express how alternative solutions contribute differently to goals.

5.2 Second Group: Knowledge Repositories

The majority of research contributions on knowledge management for security is dedicated to gathering and cataloguing bodies of knowledge in various structures and formats. The result of these efforts are the existing public sources of security knowledge. This part surveys these sources of knowledge and the way that knowledge is structured those sources.

5.2.1 Text-Based Unstructured Security Knowledge

The content of text-based knowledge sources such as books, standards, guidelines, and project documentations is amorphous and unstructured. Security textbooks such as [64, 3] typically include a wide variety of knowledge ranging from high-level concepts to technical details. Security standards such as [47] include practical security recommendations with emphasis following proven approaches on which the community of practitioners agrees on, and thus standards can serve as criteria for compliance. Security guidelines such as NIST guidelines [76, 25] have broad coverage similar to textbooks, but presented in more focused and practical format; therefore guidelines are used as ready-to-apply practical development guides. Security checklists such as SANS checklists[2] provide designers with tips and reminders about well-known pitfalls and solutions.

Design rationale is the explicit listing of decisions made during a design process and the reasons why those decisions were made [40]. Design rationale is an explanation of why an artifact or some part of an artifact is designed the way it is. Therefore, if such a document is developed about security design decisions within an organization, it may contain recurring attack patterns and descriptions of successful or defeated solutions as well as why a security solution was chosen. Codifying and cataloguing such information would be useful for future references.

5.2.2 Security Design Patterns

Design patterns encapsulate experts knowledge in form of a proven solution to common problems. Patterns are mostly structured text in form of a three-part rule which expresses a relationship between a certain context, a problem, and a solution. Design patterns are widely used for collecting and codifying object-oriented design knowledge. One of the most comprehensive catalogues of security patterns is published by the Open Group [8], in which

[2]http://www.sans.org/score/checklists.php

patterns are described based on the "Gang of four" format [22]. This pattern format describes the intent, motivation, applicability, structure, consequences, known uses, and related patterns.

In addition to proposal by Gross and Yu [26] for expressing design patterns using goal-orineted models, in the SERENITY project, a goal based approach for expressing security patterns were employed [1]. The solution and context of the patterns are elaborated both in text and graphical Secure Tropos goal models. Expressing patterns by Secure Tropos helps the designers to validate the pattern using reasoning techniques. Mouratidis et al. [62] also offer an approach for expressing and structuring security design pattern for agent systems. The patterns are expressed in text accompanied with social dependencies models expressed by i* notation. The proposed patterns express the consequences of applying the patterns explicitly.

5.2.3 Web-Based Knowledge Bases

There exist web portals that gather and catalogue an updated list of vulnerabilities discovered for specific platforms, operating systems, protocols and products. SANS top-20 annual security risks[3] provide updated lists of common vulnerabilities. The US-CERT Vulnerability Notes Database (VND)[4] is a well-known web-based security catalogue which gathers security vulnerabilities in a searchable database. VND catalogues vulnerabilities, their context, impact, and a reference to the vulnerability report. Generally, the structure of this knowledge source is limited to the information about the vulnerabilities, and solutions are not suggested for the vulnerabilities.

National Vulnerability Database (NVD)[5] contains Common Vulnerabilities and Exposures (CVE)[6] list that provides common names for publicly known information security vulnerabilities. NVD includes databases of security checklists, security related software flaws, misconfigurations, product names, and impact metrics. NVD supports the Common Vulnerability Scoring System (CVSS)[7] standard for all CVE vulnerabilities. The CVSS is an open standard for scoring the vulnerabilities by providing an open framework for communicating the characteristics and quantitative impacts of IT vulnerabilities.

5.2.4 Taxonomies and Ontologies

Ontologies for modeling and expressing security are another source of security knowledge. Mouratidis et al. [60] propose an ontology for modeling security, based on the Tropos approach. The ontology uses the notion of security constraints. Security constraints that contribute positively or negatively to the other requirements of the system are differentiated to identify conflicts between security and other (functional and non functional) requirements. Security constraints are imposed to different parts of the system, The Secure Tropos ontology also introduces the term secure entity which includes secure goals, tasks and resources of the system. A secure entity is introduced to the actor (or the system) in order to help in the achievement of a security constraint.

Massacci et al. [53] suggest an extended ontology based on i*, using the notion of ownership, delegation, permission, and trust. The theme of the work is on exploring vulnerabilities in the interface between the information system and the organization. this work proposes a language called SI*, which defines required constructs for designing secure socio-technical systems, based on the concepts of the i* modeling notation.

Taxonomies of security concepts are a common method for sharing security knowledge. Security taxonomies can be used for evaluating the comprehensiveness of the KB schemas. Avizienis et al. [49] provide a detailed taxonomy of dependability and security concepts

[3]http://www.sans.org/top20/
[4]http://www.kb.cert.org/vuls/
[5]http://nvd.nist.gov/
[6]http://cve.mitre.org/
[7]http://www.first.org/cvss/

such as dependability and security attributes, taxonomy and classes of faults, fault modes, classification of fault tolerance techniques, and verification approaches. In this taxonomy, the main threats to dependability and security are defined as failures, errors, and faults. Avizienis et al. [49] classify the main means to attain security and dependability attributes into fault prevention, fault tolerance, fault removal, fault forecasting. The proposed concepts taxonomy provides elementary fault classes which includes: 1) development faults or operational ones, 2) internal vs. external faults, 3) natural or human-made faults, 4) hardware or software faults, 5) malicious vs non-malicious faults, 6) deliberate or non-deliberate faults, 7) accidental faults or incompetence faults, 8) permanent vs. transient faults.

The authors use the combination of the fault classes to analyze the nature of faults and ways to deal with them. For example, deliberate, non-malicious, development faults result generally from trade-offs, either 1) aimed at preserving acceptable performance, at facilitating system utilization, or 2) induced by economic considerations; Malicious human-made faults exploit a vulnerability to cause harms; etc. The classification and such contributions are useful to locate the points that decisions are made during development for further analysis of some users or events.

Firesmith [21] suggests a taxonomy for security requirements. The taxonomy introduces a quality model, which defines what a quality is, how it is divided into sub-qualities, what are qualities criteria, etc. Quality requirements are defined with respect to the quality model. Safety and security are defined based on the quality model, and defensibility subfactors are described based on the sub-quality model. Defensibility problem type are divided into incidents, dangers, risks, and harms. The taxonomy categorizes harm as authorized and unauthorized harm, and harm to people, to property, to environment harm, and to service. Firesmith suggests a classification of safety and security incidents as well. A model for defining safety hazards vs. security threats is also provided.

Finally, the taxonomy decomposes the security requirements into pure security requirements, security-significant requirements, security system requirements, and security constraints. Pure security requirements specify minimum thresholds of various subfactors of security. Security-significant requirements are derived by using harm, incident, threat, and risk analysis to categorize non-security requirements in terms of their relative security risks. Security system requirements are the requirements associated with security subsystems. Finally, security constraints are engineering decisions that they would ordinarily be made during architecture development, design, implementation, integration, or testing.

5.3 Limitations of Existing Knowledge Sources

Generally, text-based knowledge content is amorphous and unstructured; therefore, browsing, retrieving, manipulating, and expanding the knowledge source is cumbersome. In addition, knowledge is represented informally and in natural language, thus designers and analysts need to interpret the content, which may lead to misinterpretation and inaccuracies. The unit of knowledge in textual contents is not fixed, and can be a sentence, a paragraph, a section or a chapter. This implies that the smallest chunk of reusable knowledge may be a whole section or chapter. The text-based knowledge units are tightly coupled, which makes it difficult to modularize the knowledge. These characteristics make the retrieval and application of the knowledge inefficient and often ineffective, because the designer needs to browse a large chunk of knowledge in order to extract the required information.

Textbooks and guidelines tend to be descriptive, abstract, and theory-oriented rather than practical. Standards address a specific context and domain, thus cannot be applied in a broad area. While checklists aim to remind the designer to follow certain recommendations and avoid pitfalls in technical details, and do not provide the designers with required knowledge for analyzing security solutions and making trade-off decision. Design rationale is internal documents of an organization, which are not well documented, and usually are incomplete and hard to maintain.

Design patterns are more structured than other text-based knowledge sources such as standards and textbooks. However, patterns still rely heavily on informal text-based presentation, because the structure of design reasoning is not systematically organized [26]. The goal-driven approach for expressing design patterns in [26] makes the reasoning structure behind design patterns explicit. However, no consideration is given to security-specific concepts such as vulnerabilities, threats, and security safeguards in the structure of the patterns. In addition, the resulting models may become large and complex, especially if one integrates multiple design patterns into one goal model to analyze and compare alternative design patterns for a problem and a procedure for browsing and extracting smaller units from the models is not provided.

The proposed patterns for agents in [62] does not consider cataloguing security mechanisms and assets that are target of attacks. Besides, the type of reasoning and analysis that the proposed structure of knowledge provides is not specified; therefore the proposed schema remains at the level a catalogue of security principles, rules, vulnerabilities, and attack pattern, without correlating the body of knowledge to trade-offs. Security and privacy patterns in the SERENITY project [1] do not consider attacks and vulnerabilities. In addition, the approach used in SERENITY project does not provide structures for expressing and analyzing trade-offs among alternative patterns.

The NFR security catalogues [14] do not consider modeling security-specific aspects such as threats, attacks, vulnerabilities, and assets. The security methods in the security operationalizations catalogue are not related to threats or vulnerabilities; therefore, the catalogue does not provide knowledge required for selecting security solutions for potential threats and vulnerabilities. The proposed schema of the KB in [6] does not consider cataloguing security mechanisms and assets, and does not provide the required structure to express the trade-offs.

6 Conclusions and Potential Research Opportunities

Existing security requirements framework have different definitions and views to security requirements. While traditionally, security is viewed as a protection mechanism, and several methods deal with security requirements as constraints on functionality, new trends for analyzing access control concern trust, ownership, and permission delegation issues; some approaches view security as an issue that arises from interactions of social actors; and some approaches consider security as an issue that always involves trade-offs. Several existing security analysis methods focus on modeling and analysis of attacks and vulnerabilities, and relating them to security requirements. The other group of security modeling frameworks focus on capture and analyze security goals and security countermeasures.

It is agreed that security requirements pose trade-offs on other objectives. However, currently, security trade-offs are made intuitively by practitioners. Current security requirements frameworks do not provide explicit means for modeling and analyzing security trade-offs at the requirements and architecture levels. It is useful to express security trade-offs explicit in order to avoid implicit and intuitive decision makings. In addition, explicit expression of trade-offs serves both as a record for decision rationale and a basis for trade-off reasoning. However, few software engineering contributions focus on analyzing security trade-offs and making rationale for trade-offs explicit.

The other challenge for security trade-off analysis is lack of an accessible and easy-to-use body of knowledge that facilitates making security trade-offs explicitly. We overviewed existing sources of security knowledge in general, and we discussed how each knowledge source may help making security trade-offs.

Lack of trade-off analysis activities in security requirements frameworks and the current knowledge challenges bring the need for a method for careful analysis of security trade-offs with a knowledge support. The subjective nature of security trade-offs, implicit effects of security mechanisms on privacy, usability, and functionality requirements, ethical issues involved with quantifying security, safety, and privacy benefits and costs, and lack of agreement on measurement methods and metrics for quality goals, make the security trade-offs analysis challenging. Deciding on security requirements and selecting an appropriate architecture that satisfies the requirements need careful analysis of trade-offs that each requirement and architectural solution imposes. With respect to state of the art and practice, current challenges, and the work done to date, potential research developments are foreseen:

- Improving the conceptual modeling notation for expressing security trade-offs explicitly, proposed in [17]. Criteria for such a notation need to be gathered and justified as well.
- Developing a trade-off analysis method that facilitates the decision making process. Both quantitative and qualitative methods and the possibility to combine qualitative and quantitative measures can be explored.
- Designing a process model for intertwining the trade-off analysis to security requirements and architecture design activities.
- Empirical studies such ethnographies, interviews, and surveys to understand how trade-offs are made in current practices, how software developers model and analyze security requirements, and how security requirements affect the architecture design.
- Empirical experiments such as controlled experiments to evaluate usefulness, expressiveness, and understandability of the proposed goal-oriented security conceptual technique.
- Designing and populating a knowledge base for structuring security trade-offs knowledge. To develop such a knowledge base, the required knowledge items for making security trade-offs need to be examined.
- Experiments to understand the psychology of security trade-off analysis and decision making in current software engineering state of the practice.

References

[1] Serenity project, a1.d3.1 - initial set of security and privacy patterns at organizational level. 2007.

[2] I. F. Alexander. Initial industrial experience of misuse cases in trade-off analysis. In *RE '02: Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, pages 61–70, Washington, DC, USA, 2002. IEEE Computer Society.

[3] R. Anderson. *Security Engineering: a guide to Building dependable Distributed systems.* John Wiley and Sons, 2001.

[4] Y. Asnar and P. Giorgini. Modelling risk and identifying countermeasure in organizations. pages 55–66, Samos, Greece, August 2006. LNCS 4347.

[5] Y. Asnar, R. Moretti, M. Sebastianis, and N. Zannone. Risk as Dependability Metrics for the Evaluation of Business Solutions: A Model-driven Approach. In *Proc. of DAWAM'08*, pages 1240–1248. IEEE Press, 2008.

[6] S. Barnum and G. McGraw. Knowledge for software security. *IEEE Security and Privacy*, 3(2):74–78, 2005.

[7] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice.* Second Edition, Addison Wesley, Boston, MA, USA, 2003.

[8] B. Blakley and C. Heath. Security design patterns. 2004.

[9] B. Blakley, E. McDermott, and D. Geer. Information security is information risk management. In *NSPW '01: Proceedings of the 2001 workshop on New security paradigms*, pages 97–104, New York, NY, USA, 2001. ACM.

[10] F. Braber, I. Hogganvik, M. S. Lund, K. Stolen, and F. Vraalsen. Model-based security analysis in seven steps — a guided tour to the coras method. *BT Technology Journal*, 25(1):101–117, 2007.

[11] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. *JAAMAS*, 8(3):203–236, 2004.

[12] P. Bresciani, P. Giorgini, and H. Mouratidis. On security requirements analysis for multi-agent systems. In *Proc. of 2nd Int. Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS)*, pages 35–48, 2003.

[13] S. A. Butler. Security attribute evaluation method: a cost-benefit approach. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 232–240, New York, NY, USA, 2002. ACM.

[14] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering.* Springer, October 1999.

[15] F. den Braber, T. Dimitrakos, B. A. Gran, M. S. Lund, K. Stolen, and J. O. Aagedal. The coras methodology: model-based risk assessment using uml and up. pages 332–357, 2003.

[16] P. T. Devanbu and S. Stubblebine. Software engineering for security: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 227–239, New York, NY, USA, 2000. ACM.

[17] G. Elahi and E. Yu. A goal oriented approach for modeling and analyzing security trade-offs. In *Proc. of ER'07*, LNCS 4801, pages 375–390. Springer, 2007.

[18] S. Evans, D. Heinbuch, E. Kyule, J. Piorkowski, and J. Wallner. Risk-based systems security engineering: Stopping attacks with intention. *IEEE Security and Privacy*, 2(6):59–62, 2004.

[19] S. Evans and J. Wallner. Risk-based security engineering through the eyes of the adversary. *In Information Assurance Workshop, IAW '05.*, pages 158–165, 2005.

[20] D. Firesmith. Security use cases. *Journal of Object Technology*, 2(1):53–64, 2003.

[21] D. Firesmith. A taxonomy of security-related requirements. In *Int. Workshop on High Assurance Systems, RHAS'05.* 2005.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Professional, 1995.

[23] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modeling security requirements through ownership, permission and delegation. In *In Proc. of RE'05*, pages 167–176. IEEE Computer Society, 2005.

[24] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Formal reasoning techniques for goal models. *Journal of Data Semantics*, 1:1–20, 2003.

[25] T. Grance, M. Stevens, and M. Myers. Guide to selecting information technology security products. *Recommendations of the National Institute of Standards and Technology, NIST Special Publication 800-36*, 2003.

[26] D. Gross and E. Yu. From non-functional requirements to design through patterns. *Requirements Engineering Journal*, 6:18–36, 2001.

[27] S. Gupta and J. Winstead. Using attack graphs to design systems. *IEEE Security and Privacy*, 5(4):80–83, 2007.

[28] B. Haley, C. Laney, D. Moffett, and B. Nuseibeh. Using trust assumptions with security requirements. *Requir. Eng.*, 11(2):138–151, 2006.

[29] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. *TSE*, 34(1):133–153, 2008.

[30] C. B. Haley, R. C. Laney, J. D. Moffett, and B. Nuseibeh. The effect of trust assumptions on the elaboration of security requirements. In *RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International*, pages 102–111, Washington, DC, USA, 2004. IEEE Computer Society.

[31] C. B. Haley, R. C. Laney, and B. Nuseibeh. Deriving security requirements from crosscutting threat descriptions. In *Proc of AOSD'04*, pages 112–121. ACM, 2004.

[32] G. Helmer, J. Wong, . M. Slagell, . V. Honavar, . L. M. , Y. Wang, and R. L. . Software fault tree and colored petri net based specification, design and implementation of agent-based intrusion detection systems. *IEEE Transactions of Software Engineering*, 7:2002, 2002.

[33] G. Helmer, J. Wong, M. Slagell, V. Honavar, and L. Miller. A software fault tree approach to requirements analysis of an intrusion detection system. In *Requirements Engineering Journal*, pages 207–220, 2001.

[34] S. H. Houmb, F. D. Braber, M. S. Lund, K. Stlen, and S. T. Informatics. Towards a uml profile for model-based risk assessment. In *In UML2002, Satellite Workshop on Critical Systems Development with UML*, pages 79–91, 2002.

[35] S. H. Houmb and G. Georg. The aspect-oriented risk-driven development (aordd) framework. In *In Proceedings of the International Conference on Software Development (SWDC-REX)*, pages 81–91, 2005.

[36] S. H. Houmb, J. Jrjens, G. Georg, and R. France. An integrated security verification and security solution trade-off analysis. *In Integrating Security and Software Engineering: Advances and Future Vision. Mouratidis, H. and Giorgini, P. (eds). Idea Group Inc.*, 2006.

[37] ISO/IEC. Risk management-vocabulary-guidelines for use in standards. ISO/IEC Guide 73, 2002.

[38] ISO/IEC. Management of Information and Communication Technology Security – Part 1: Concepts and Models for Information and Communication Technology Security Management. ISO/IEC 13335, 2004.

[39] M. Jackson. *Problem frames: analyzing and structuring software development problems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[40] A. Jarczyk, P. Loffler, and F. Shipmann. Design rationale for software engineering: a survey. *In Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, pages 577–586, 1992.

[41] P. Johnson, R. Lagerstrom, P. Norman, and M. Simonsson. Extended influence diagrams for enterprise architecture analysis. *In: Enterprise Distributed Object Computing Conference, EDOC '06. 10th IEEE Int*, pages 3–12, 2006.

[42] B. N. Jonathan D. Moffett, Charles B. Haley. Core security requirements artefacts, the open university, department of computing. Technical Report 2004/23, 2004.

[43] J. Jürjens. Umlsec: Extending uml for secure systems development. In *Proc of UML'02*, pages 412–425. Springer, 2002.

[44] J. Jürjens. Using umlsec and goal trees for secure systems development. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 1026–1030, New York, NY, USA, 2002. ACM.

[45] D. Kahneman and A. Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–292, 1979.

[46] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere. The architecture tradeoff analysis method. *iceccs*, 00:0068, 1998.

[47] M. J. Kenning. Security management standard — iso 17799/bs 7799. *BT Technology Journal*, 19(3):132–136, 2001.

[48] J. Krogstie, A. L. Opdahl, and S. Brinkkemper. Capturing dependability threats in conceptual modelling. *Conceptual Modelling in Information Systems Engineering*, pages 247–260, 2007.

[49] J.-C. Laprie and B. Randell. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, 2004. Fellow-Algirdas Avizienis and Senior Member-Carl Landwehr.

[50] L. Liu, E. Yu, and J. Mylopoulos. Analyzing security requirements as relationships among strategic actors. In *Proc. of SREIS'02*, October 2002.

[51] L. Liu, E. Yu, and J. Mylopoulos. Security and privacy requirements analysis within a social setting. In *Proc. of RE'03*, page 151. IEEE Computer Society, 2003.

[52] T. Lodderstedt, D. A. Basin, and J. Doser. Secureuml: A uml-based modeling language for model-driven security. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 426–441, London, UK, 2002. Springer-Verlag.

[53] F. Massacci, J. Mylopoulos, and N. Zannone. An Ontology for Secure Socio-Technical Systems. In *Handbook of Ontologies for Business Interaction.* 2007. To appear.

[54] R. Matulevicius, N. Mayer, H. Mouratidis, E. Dubois, P. Heymans, and N. Genon. Adapting secure tropos for security risk management in the early phases of information systems development. In *CAiSE*, pages 541–555, 2008.

[55] J. McDermott and C. Fox. Using abuse case models for security requirements analysis. In *Proc of ACSAC'99*, page 55. IEEE Computer Society, 1999.

[56] J. P. McDermott. Attack net penetration testing. In *Proc. of NSPW'00*, pages 15–21. ACM, 2000.

[57] N. R. Mead and G. McGraw. A portal for software security. *IEEE Security and Privacy*, 3(4):75–79, 2005.

[58] N. Meyer, A. Rifaut, and E. Dubois. Towards a Risk-Based Security Requirements Engineering Framework. REFSQ-Proc. Of Internat. *Workshop on Requirements Engineering for Software Quality*, 2005.

[59] H. Mouratidis, P. Giorgini, and G. Manson. Modelling secure multiagent systems. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 859–866, New York, NY, USA, 2003. ACM.

[60] H. Mouratidis, P. Giorgini, and G. Manson. An ontology for modelling security: The tropos approach. In *KES*, pages 1387–1394, 2003.

[61] H. Mouratidis, P. Giorgini, G. Manson, and I. Philp. A natural extension of tropos methodology for modelling security. In *Proceedings of the Workshop on Agent-oriented methodologies, at OOPSLA*, 2002.

[62] H. Mouratidis, P. Giorgini, and M. Schumacher. Security patterns for agent systems. In *Proceedings of the Eight European Conference on Pattern Languages of Programs (EuroPLoP), Irsee*, 2003.

[63] B. Nuseibeh. Weaving together requirements and architectures. *Computer*, 34(3):115–117, 2001.

[64] C. P. Pfleeger and S. L. Pfleeger. *Security in Computing.* Prentice Hall Professional Technical Reference, 2002.

[65] C. Phillips and L. P. Swiler. A graph-based system for network-vulnerability analysis. In *Proc. of NSPW'98*, pages 71–79. ACM, 1998.

[66] K. Pohl and E. Sikora. The co-development of system requirements and functional architecture. pages 229–246. 2007.

[67] W. N. Robinson, S. D. Pawlowski, and V. Volkov. Requirements interaction management. *ACM Comput. Surv.*, 35(2):132–190, 2003.

[68] L. Rstad. An extended misuse case notation: Including vulnerabilities and the insider threat. In *The Twelfth Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'06)*, 2006.

[69] R. Sandhu. Good-enough security: Toward a pragmatic business-driven discipline. *IEEE Internet Computing*, 7(1):66–68, 2003.

[70] B. Schneier. Attack trees. *Dr. Dobb's Journal*, 24(12):21–29, 1999.

[71] B. Schneier. *Beyond Fear.* Springer, 2003.

[72] B. Schneier. The psychology of security. *Commun. ACM*, 50(5):128, 2007.

[73] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated generation and analysis of attack graphs. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 273, Washington, DC, USA, 2002. IEEE Computer Society.

[74] G. Sindre and A. L. Opdahl. Templates for misuse case description. In *Proceedings of the 7 th International Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ'2001*, pages 4–5, 2001.

[75] G. Sindre and L. Opdahl. Eliciting security requirements with misuse cases. *Requir. Eng.*, 10(1):34–44, 2005.

[76] M. Tracy, W. Jansen, K. Scarfone, and J. Butterfield. Guidelines on electronic mail security. *Recommendations of the National Institute of Standards and Technology, NIST Special Publication 800-45*, 2007.

[77] A. van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proc. of ICSE'04*, pages 148–157. IEEE Computer Society, 2004.

[78] A. van Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. Softw. Eng.*, 26(10):978–1005, 2000.

[79] W. E. Vesely, F. F. Goldberg, N. Roberts, and D. F. Haasl. Fault tree handbook. Technical Report NUREG-0492, U.S. Nuclear Regulatory Commission, January 1981.

[80] J. Viega, T. Kohno, and B. Potter. Trust (and mistrust) in secure applications. *Commun. ACM*, 44(2):31–36, 2001.

[81] E. Yu. *Modeling Strategic Relationships for Process Reengineering.* PhD thesis, Department of Computer Science, University of Toronto, Canada, 1995.

[82] N. Zannone. The si* modeling framework: Metamodel and applications. *To appear in International Journal of Software Engineering and Knowledge Engineering*, 2008.