Characterizations and Boundaries of Security Requirements Patterns

Rocky Slavin, Hui Shen, and Jianwei Niu
Department of Computer Science
University of Texas at San Antonio, San Antonio, Texas, USA
{*rslavin, hshen, niu*}*@cs.utsa.edu*

***Abstract*—Very often in the software development life cycle, security is applied too late or important security aspects are overlooked. Although the use of security patterns is gaining popularity, the current state of security requirements patterns is such that there is not much in terms of a defining structure. To address this issue, we are working towards defining the important characteristics as well as the boundaries for security requirements patterns in order to make them more effective. By examining an existing general pattern format that describes how security patterns should be structured and comparing it to existing security requirements patterns, we are deriving characterizations and boundaries for security requirements patterns. From these attributes, we propose a defining format. We hope that these can reduce user effort in elicitation and specification of security requirements patterns.**

***Keywords*-security; patterns; security requirements**

I. Introduction

The design pattern has become an important mechanism in the software development process. By providing a solution to recurring problems, the design pattern serves to streamline and speed up the development process. Further, since design patterns have a general structure, they serve as a way to easily transfer knowledge or experience among software engineers. This way of keeping ideas and solutions together has also served as a way for novice software engineers to inherit common techniques and best practice methods.

Though security design patterns have become well used mechanisms to address security during the design process, the lack of security requirements specification has hindered the ability to properly integrate security from the start. Very often, security requirements or policies have to be applied after-the-fact by way of patches or solutions that are applied so late that effectiveness is sacrificed [1]. To alleviate this problem, Yoder and Barcalow were the first to make use of security patterns. [2].

In defining a problem, context and solution, the design pattern has a standard structure. This is inspiration for expressing security requirements in terms of patterns. By understanding the objectives for security requirements patterns and studying the ways that existing requirements patterns are presented, a standardized way to explicitly represent requirements using patterns can be derived. Our approach begins by finding common characteristics in existing patterns and comparing them with the guidelines established by existing pattern formats for general security patterns.

Discussion on the state of security patterns in general has shown some common characteristics in existing patterns. Schumacher et al. use a common structure that is widely used to describe security patterns [3]. Patterns following this structure tend to be design patterns. However, the general pattern format may be a starting point for defining the boundaries of security requirements patterns. Our research suggests that some of the characteristics of the format may be necessary, but they are not sufficient to describe a security requirements patterns. By finding the common traits of existing security requirements patterns and comparing them with what a security pattern should include the boundary of these patterns can be identified. A survey of the existing security patterns conducted by Yoshioka et al. provides a basis for our study [4]. Inspired by grounded theory [5], and in order to analyze current patterns and understand them better, we have compared existing patterns to deduce frequently used characteristics [5]. Through this practice, we can develop the boundaries of security requirements patterns from existing patterns.

Even with security in mind, the use of patterns can be hindered by applying the wrong ideas. Firesmith outlines the components of security requirements, addressing the confusion between these requirements and the mechanisms used to fulfill them [6]. In specifying mechanisms, such as the use of cryptography or user identifiers, requirements engineering is hindered by overlapping with architectural design. This also restricts the generic nature of the requirements pattern. It is important to correctly address security requirements at the right stage of software development. We believe that through the creation and use of security requirements patterns, requirements engineers and software practitioners can more effectively treat security concerns.

Security requirements patterns should also focus on some common high-level security objectives as well as possible threats and attacks. In order to maintain these basic security objectives, certain guidelines should be used in security requirements patterns. Firesmith argues that in order to properly preserve many requirements such as authorization, integrity, privacy, immunity and more, guidelines such as the general use of security policy, understanding and maintaining feasibility, correctness requirements and the use of models such as misuse cases should be followed [6]. We believe that the use of these kinds of guidelines, aside from

978-1-4673-4376-3/12/$31.00 © 2012 IEEE

a pattern format, is necessary for eliciting and describing requirements.

The specification of requirements often involves specification of external requirements. The inclusion of context-specific requirements is necessary for many systems. Very often, regulations or laws such as HIPAA [7] or PCI [8] must be complied with. Similarly, context can vary widely depending on what the software is being created for. Since a pattern is created as a generic way to solve a recurring problem, we address whether or not security requirements patterns should include these external forces explicitly, or if they should be independent. Some papers suggest integrating external guidelines directly into security requirements patterns. For example, Compagna et al. [9], suggests using guidelines such as those from the European Directive on Data Protection [10] in order to develop security patterns for requirements engineering. We believe that, depending on the context of the requirement, security requirements patterns should integrate corresponding external regulations when possible. By doing so, patterns can be extracted from the regulations themselves.

II. Literature Review

The idea of integrating security into software patterns is not completely new. The importance of security in software systems has been a cause for recurring security concerns which have lead to much research and discussion on the topic. As noted in [4], it is difficult to adapt security patterns to each phase of the software life cycle. For this reason, classification of security patterns by their respective software development phase is useful in their review and critique.

Existing work on the state of security patterns in general does exist. In [3], Schumacher et al. directly define a security pattern as:

> A particular recurring security problem that arises in specific contexts, and presents a well-proven generic solution for it. The solution consists of a set of interacting roles that can be arranged into multiple concrete design structures, as well as a process to create one particular such structure.

This paper also establishes and follows a formal outline, or "General Pattern Format" [3], in describing patterns. This format includes the following:

- *Name*
- *Also Known As*
- *Example*
- *Context* - The situations in which the pattern may apply.
- *Problem* - The problem that the pattern addresses, including a discussion of its associated forces.
- *Solution* - The fundamental solution principle underlying the pattern.
- *Structure* - A detailed specification of the structural aspects of the pattern, using appropriate notations.
- *Dynamics* - Typical scenarios describing the run-time behavior of the pattern.
- *Implementation* - Guidelines for implementing the pattern.
- *Example Resolved* - Discussion of any important aspects for resolving the example that are not otherwise covered.
- *Variants* - A brief description of variants or specializations of a pattern.
- *Known Uses*
- *Consequences*
- *See Also*

The authors describe seven security design patterns using this method, many of which are described as extensions of one another. This format, or similar ones, is very common to security design patterns. In fact, Gamma et al. describe the three essential parts of a design pattern as the structure in terms of context, an abstract description of objects and the resulting consequences [11]. The important points in the General Pattern Format, such as context, problem, solution, structure and consequences, are directly related to the description of design patterns by Gamma. In reviewing the following papers, we hoped to understand whether these guidelines established by [3] were necessary in describing security requirements patterns and, further, whether they were sufficient.

It is important to note that the authors of these papers do not necessarily attempt to follow the General Pattern Format. We intend to use the General Pattern Format as a starting point for understanding how the patterns were described and what should be included in order to better convey the pattern structure.

We believe that, as a higher-level intent in terms of the requirement, security requirements patterns should also focus on fundamental security objectives. For their uses in this paper, we define the following security objectives:

- *Confidentiality* - Information is only disclosed to those authorized.
- *Integrity* - Information is only altered with respect to the desires of the owner.
- *Availability* - Information is accessible when needed.
- *Accountability* - Changes in information can be traced to the actors responsible.
- *Least Privilege* - Users and programs should operate using the least set of privileges necessary to complete their corresponding jobs. [12]

These objectives appear in terms of goals and threat preventatives.

Literature on patterns for the requirements phase is not quite at the same state as that which pertains to the design phase. Many of the papers for this stage make note of this. We first present existing security design patterns as a way to see how the more established pattern type is described, followed by existing security requirements patterns.

A. Existing Security Design Patterns

In a very similar style to the General Pattern Format presented in [3], [2] presents seven classic design patterns which deal with requirements by allowing the details of security to be addressed later in software development. These security patterns are meant to be applied early in the software lifecycle. The authors warn of the danger and difficulty in attempting to retrofit a system with security patterns after security has become a problem.

UML is used as a primary tool in conjunction with the previously mentioned format in [13]. In this paper, the authors discuss a pattern language consisting of three basic security design patterns. The authors suggest dividing a software system based on an architectural pattern consisting of layers and applying the described security patterns to individual levels of this hierarchy in order to address higher level security.

In some cases, papers were concerned with very specific contexts. [14] addresses security issues for voice over IP connections through the use of patterns. In this paper, the issues are very specific problems that, while the author considers them security patterns, the value of a solution to a generic problem is lost. While some of these patterns can be applied to recurring problems others, such as the Signed Authenticated Call pattern, attend to very specific problems such as "How can an attacker be prevented from masquerading as a VoIP terminal device, either IP or standard, when network subscribers want to establish a voice call?".

The modification of an existing pattern to apply security is discussed in [15]. Here, the authors take the existing Broker pattern and show, through the use of UML, how existing issues with the pattern can be addressed by assessing security responsibilities as requirements and applying solutions to the existing pattern.

B. Existing Security Requirements Patterns

Table I describes some notable patterns from papers discussed in this section.

In a paper regarding the proposition of integrating legal requirements into requirements engineering, the lack of tools for validation of security patterns is noted [9]. Compagna et al. describe the current lack of reference to legal theory by patterns designed for systems which must be conscientious of these laws. The paper describes four patterns as well as a pattern design system through the use of graphical SI* modeling diagrams and the Secure Tropos methodology. The use of the SI* modeling framework to model the goals and assets of stakeholders illustrates the goal-oriented approach used for these requirements patterns. Similarly to the Schumacher's general pattern format, this paper lists context, requirement, solution and consequences.

Kis goes further in analyzing the current state of security requirements patterns by presenting and analyzing two requirements antipatterns [16]. These antipatterns are patterns which, while initially seeming useful, have failed to adequately protect the software system. The antipatterns, and their corresponding alternative secure patterns, are described in terms of their problem, background, context, forces, faulty beliefs, antipattern solution, consequences, symptoms and refactored solution. Through this analysis, it is found that it is vital, in the development of security requirements patterns, for not only the threat or goal to be clear, but the consequences to be factored into the solution.

Mouratidis et al. also provide a goal-oriented approach using the Tropos methodology to model their patterns [17]. By using Tropos, patterns can be described in terms of agents and their goals. The paper attempts to merge this agent oriented software engineering approach with patterns in order to create a standardized pattern language.

Problem Frames [20] are used in [18] as a means to describe threats. By graphically representing the dependencies between preconditions and their corresponding postconditions, Hatebur et al. describe a way to analyze a security problem in order to reach a set of security requirements and approaches to handle those requirements

In [19], Schumacher asks the question "what makes a security pattern a security pattern?" He goes on to define a security pattern template which elaborates on the general pattern elements such as problem, context and solution and goes further by discussing the idea of standardizing this template based on the Common Criteria. The paper relates environmental assumptions and policy statements to context; security objectives, threats and attacks to the problem; and functional security requirements to the forces. These elements are a direct part of requirements engineering and are valuable to the structure of security requirements patterns. By following the standardized statements provided by the Common Criteria, a security requirements pattern can be more clearly defined. The patterns described do not strictly pertain to the requirements stage, but this relation between the pattern structures and the Common Criteria provides some insight into how requirements patterns should be formatted.

While this is not a complete review, the patterns and their corresponding papers listed provide enough insight to see that design patterns currently have a better, more established structure. The similar structures used to present security design patterns in [19], [3], [2], [13], [14] and [21] more clearly convey the problem, context and solutions than the various styles of presentation used for security requirements patterns.

III. Observations

The existing security requirements patterns that were reviewed clearly showed a need for either a goal or a threat, but did not show common boundaries. Some of these papers, such as [17] and [9], described requirements patterns as methodologies used to generate and identify forces and the

Table I
SECURITY REQUIREMENTS PATTERNS

Pattern Name	Requirements	Context	Notation	Goal/Threat
Perimeter security: the Maginot line of enterprise applications [16]	The security of a typical n-tier enterprise application needs to consider time to market and the difficulty with applying general system security theory in software development.	Modern intranet infrastructure	Natural language	Threat
Agent Authenticator [17]	Many malicious agents will try to masquerade their identity when requesting access to an agency. Agents should be authenticated as they enter the agency.	N/A	Tropos	Goal
SPF: Anonymity [18]	Spy cannot deduce relation between Subject and AnService using AccessTarget.	N/A	Problem Frame	Threat
Pseudonymous email [19]	Despite the desire for anonymity users may be required to authenticate to the email service. Software engineers need to define how to use an email service without revealing own's identity.	Web based system	Natural language	Goal
Access Control [9]	The Data Requester shall access information if he is an authorized actor.	Data intensive system	Secure Tropos	Goal
Non-repudiation pattern in absence of trust [9]	The Delegator shall have evidence that the Executor cannot repudiate his commitment.	Data intensive system	Secure Tropos	Goal

relations among those forces. Others described patterns similarly to the way design patterns are presented, by attempting to provide a solution based on the existing forces.

A. Requirements and Solutions

Security design patterns are concerned with a solution as a purpose for a pattern. For security requirements patterns we believe that there should not be a specific mechanism that addresses the goal or threat. By creating a direct solution, the generic purpose of the pattern would be limited by the mechanism used to address the requirements. Further, the specification of explicit solutions limits architects by restricting them to the security mechanisms specified by the requirements [6]. In lieu of a solution, the structure presented by the pattern in terms of threats and goals represented by models should guide the requirements engineer to a position to approach the next phase in software development with security requirements addressed.

According to the guidelines for security requirements put forth by [6], "security requirements are driven by threats." While we do not disagree, we believe that security requirements patterns can focus on a goal in order to combat threats. We found that nearly all security patterns were either concerned with higher level threats or direct goals. These representations of threats and goals provide a basis for patterns. Similarly to how a security design pattern may be concerned with a problem, this common characteristic and the foundation for the pattern that it provides shows that a security requirements pattern should start with a threat or goal.

B. Modeling Languages and Methodologies for Goal-Oriented Patterns

Many of the reviewed patterns specified requirements as goals. The use of modeling languages and methodologies such as i* [22] with Tropos [23] and SI* [24] with Secure Tropos provide a way to specify structures in patterns to protect the high-level security objectives defined as goals.

The i* framework models the relations between the system and its environment. The framework consists of two models, the Strategic Dependency model, which describes the dependency among multiple actors; and the Strategic Rationale, which describes the actor concerns. Each actor may have goals to capture its high-level objectives. By depending on other actors, a single actor may be able to achieve goals which would normally be difficult. Often used with i*, Tropos is a methodology which supports design and analysis of the software development process and is used to build agent-oriented software systems. The SI* modeling language integrates the system secure needs into requirement model represented using i*. SI* employs the major concepts of i*, such as actor and goal. It also employs the notion of permission delegation, execution dependency, trust of permission and trust of execution to model security policies.

C. Modeling Languages for Threat-Oriented Patterns

Threats are possible scenarios which jeopardize high-level security objectives. In the same way that goal-oriented patterns focus on goals to achieve these security objectives, threat-oriented patterns use requirements as a preventative to actively protect the objective against dangers. Similarly to the use of i* and Tropos, the use of problem frames, attack trees or misuse and abuse cases were commonly used to model threats and vulnerabilities for requirements.

Problem frames are introduced to define identifiable problems in terms of their context, characteristics of their domain, interface and requirements [20]. They are described using frame diagrams. These consist of rectangles, denoting the domains and requirements, and the links between rectangles, denoting the interfaces. The use of these problem frames help to analyze the threat or potential problems regarding requirements specifications.

Attack Trees formally describe the possible ways in which a system can be attacked [25]. In an attack tree, the root node represents a goal. The children nodes refine the root node with possible attacks while the leaf nodes represent the attacks which cannot be refined any more. The attack tree contains AND nodes and OR nodes. AND nodes denote different steps in achieving a goal. OR nodes denote alterna-

tive ways to achieving the same goal. The leaf nodes can be assigned values, which are boolean or continuous, once the tree is created. The value of other nodes can be calculated from the children nodes. Multiple values can be assigned to the same node to represent different combined variables. Through the use of attack trees, knowledge of security can be reused. This idea is important when concerning patterns.

Alexander describes the implementation of misuse cases as a way to model threats in a similar way to traditional use cases [26]. Misuse cases, which are also known as abuse cases, are the negative form of use cases. Each misuse case represents a group of scenarios of possible threats. For example, a normal use case involving a database system may involve a client accessing information to which they have the appropriate privileges, while a misuse case may describe an unauthorized actor, such as a hacker, accessing data which they should not be able to retrieve. These misuse cases can be part of development from a system to its subsystem recursively. Through this method, the nonfunctional requirements that can be overlooked can be addressed at any system level and security requirements patterns can be applied with these threats in mind.

IV. Standardized Pattern Format

A common structure among security requirements patterns was not apparent. While some papers, such as [16], discussed patterns in a direct format similar to the General Pattern Format presented by Schumacher [3], most used different styles of presentation to describe their patterns. In papers which did not use a similar format, the use of natural language was complimented by the previously mentioned modeling languages.

A. Pattern Format

The work by Schumacher et al. on security patterns as well as the guidelines on requirements organized by Firesmith set a foundation for security requirements patterns that can be built upon in order to establish a well-defined structure. We believe that it is important to have a general structure in order to increase the quality of the pattern as well as the ability to elicit them. Based on the reviewed patterns we recommend the following modified pattern format:

- *Name*
- *Context* - The situations in which the pattern may apply.
- *Requirement* - Refined from high-level security goals to detailed non-functional requirements.
  - *Goals*
  - *Threats/Attacks*
  - *Regulations*
- *Structure* - A generic organization of the pattern.
- *Consequences* - Potential constraints on other requirements.
- *Also Known As - optional*
- *Example - optional*
- *Known Uses - optional*
- *See Also - optional*

In place of a problem we list requirement. Since requirements should be concerned with high-level security objectives, they should be specified and structured through the use of a modeling language. We also believe that the specification of a solution is not necessary in a security requirements pattern and should be replaced by the structure. The organizational structure results through the use of modeling to understand and resolve requirements. By modeling the requirement itself with all relevant forces the resulting organizational structure should take the place of a solution.

In specifying security requirements, not all consequences are security related. Often, constraints created by security requirements affect other functional, non-security non-functional and security non-functional requirements [3]. For this reason, we believe that consequences are an important aspect of requirements specification and serve an important purpose in the structure of security requirements patterns.

Since patterns have traditionally served as a means for communication between software engineers, clarity and standardization is important. Certainly, the modeling techniques utilized by pattern developers provide comprehensibility, but a common standardization can help developers to recognize and gain the full benefits of security requirements patterns. Further, this format provides a structure which allows for easier pattern elicitation.

B. Case Study

We have begun to evaluate our pattern format by eliciting and deriving patterns from HIPAA regulations. The first step is eliciting high-level security objectives as requirements from the regulations of the health care context. While this case study is not yet complete, our hopes are to show that the core of a security requirements pattern, the requirement itself, should focus on the fundamental security objectives as well as to validate the structure we propose.

The HIPAA (Health Insurance Portability and Accountability Act) [7] regulates the transmission and use of confidential health information, which are referred as protected health information (PHI) among covered entities. Covered entities are the organizations affected by HIPAA, including hospitals, insurance companies, doctors and so on. Let us look at two HIPAA rules regarding PHI. Each rule contains the basic elements of our pattern guideline: name, requirement in terms of high-level security objectives and context. By extracting these elements and creating organization of threats or goals for the specified context, the foundation of a security pattern can be elicited.

HIPAA §164.508(a)(2) Notwithstanding any provision of this subpart, other than the transition provisions in §164.532, a covered entity must obtain an authorization for any use or disclosure of psychotherapy notes, except: (i) To carry out the following treatment, payment, or health care operations: (A) Use by the originator of the psychotherapy notes for

treatment; (B) Use or disclosure by the covered entity for its own training programs in which students, trainees, or practitioners in mental health learn under supervision to practice or improve their skills in group, joint, family, or individual counseling; or (C) Use or disclosure by the covered entity to defend itself in a legal action or other proceeding brought by the individual; and (ii) A use or disclosure that is required by §164.502(a)(2)(ii) or permitted by §164.512(a); §164.512(d) with respect to the oversight of the originator of the psychotherapy notes; §164.512(g)(1); or §164.512(j)(1)(i).

HIPAA §164.502(b)(1) When using or disclosing protected health information or when requesting protected health information from another covered entity, a covered entity must make reasonable efforts to limit protected health information to the minimum necessary to accomplish the intended purpose of the use, disclosure, or request.

These policies regulate different security objectives as requirements. §164.508(a)(2) regulates the authorization of using or disclosing a specific type of PHI, psychotherapy notes. §164.502(b)(1) regulates the least privilege of using or disclosing PHI. We will continue our case study to evaluate the effectiveness of our proposed pattern. In extracting a pattern from these regulations, we hope to show that our pattern format reduces user efforts in elicitation of security patterns from existing regulations.

V. Concluding Remarks and Future Work

Since the use of security requirements patterns is still a new practice, now is the time for a standardized format and good understanding of what a security requirements pattern should include to be created. By comparing existing security requirements patterns to various ideals on security and the ideas behind patterns, we hope that our resulting security requirements pattern format illuminates the current state of security requirements patterns and enables pattern developers to discover and specify new security requirements patterns in terms of our guidelines.

In the future, we will continue our work on formalizing security requirements patterns in order to provide pattern developers with a better way to elicit and extract patterns.

Acknowledgement

Jianwei Niu is supported in part by NSF award CNS-0964710 and NSA-CMU subcontract 1130145-280847.

References

[1] P. T. Devanbu and S. Stubblebine, "Software engineering for security: a roadmap," in *ICSE*, 2000, pp. 227–239.

[2] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security," in *PLoP*, 1997, pp. 1–37.

[3] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons Inc, 2006.

[4] N. Yoshioka, H. Washizaki, and K. Maruyama, "A survey on security patterns," *Progress in Informatics*, vol. 5, pp. 35–47, 2008.

[5] A. L. S. Barney G. Glaser, *The Discovery of Grounded Theory: Strategies for Qualitive Research*. Aldine Transaction, 1967.

[6] D. G. Firesmith, "Engineering security requirements," *Journal of Object Technology*, vol. 2, no. 1, pp. 53–68, 2003.

[7] P. L. 104-191, *Health Insurance Portability and Accountability Act of 1996*, 104th Congress, 1996.

[8] S. S. Council, *Payment Card Industry (PCI) Data Security Standard*, 2nd ed., 2010.

[9] L. Compagna, P. El Khoury, A. Krausová, F. Massacci, and N. Zannone, "How to integrate legal requirements into a requirements engineering methodology for the development of security and privacy patterns," *Artif. Intell. Law*, vol. 17, no. 1, pp. 1–30, 2009.

[10] E. Commission, *Directive 95/46/ec on the protection of individuals with regard to the processing of personal data and on the free movement of such data*, 1995.

[11] E. Gamma, R. Helm, R. E. Johnson, and J. M. Vlissides, "Design patterns: Abstraction and reuse of object-oriented design," in *ECOOP*, 1993, pp. 406–431.

[12] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," in *Proceedings of the fourth ACM Symposium on Operating System Principles*, 1975.

[13] E. B. Fernandez and R. Pan, "A pattern language for security models," in *PloP*, 2001.

[14] E. B. Fernandez, J. C. Pelaez, and M. M. Larrondo-Petrie, "Security patterns for voice over ip networks," in *ICCGI*, 2007, pp. 33–38.

[15] P. Morrison and E.B.Fernandez, "Securing the broker pattern," in *EuroPLoP*, 2006.

[16] M. Kis, "Information security antipatterns in software requirements engineering," in *PLoP*, 2002.

[17] H. Mouratidis, M. Weiss, and P. Giorgini, "Modeling secure systems using an agent-oriented approach and security patterns," *Patterns, Inter. Jour. on Soft. Engineering and Knowledge Engineering*, vol. 16, no. 3, pp. 471–498, 2006.

[18] D. Hatebur, M. Heisel, and H. Schmidt, "A security engineering process based on patterns," in *DEXA*, 2007, pp. 734–738.

[19] M. Schumacher, "Security patterns and security standards - with selected security patterns for anonymity and privacy," in *EuroPLoP*, 2003.

[20] M. Jackson, *Problem frames: analyzing and structuring software development problems*. Addison-Wesley, 2001.

[21] S. Romanosky, A. Acquisti, J. Hong, L. F. Cranor, and B. Friedman, "Privacy patterns for online interactions," in *PLoP*, 2006, pp. 1–9.

[22] E. S. K. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *RE*, 1997, pp. 226–235.

[23] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[24] F. Masacci, J. Mylopoulos, and N. Zannone, "An ontology for secure socio-technical systems," in *Handbook of ontologies for business interaction*, 2007, pp. 188–207.

[25] B. Schneier, "Attack trees: Modeling security threats," *Dr. Dobb's Journal*, 1999.

[26] I. Alexander, "Misuse cases: use cases with hostile intent," in *IEEE Software*, vol. 20, 2003, pp. 58–66.