Una Taxonomía de Requerimientos de Seguridad de Software

A Taxonomy of Software Security Requirements

Marta E. Calderón C., MSc.
Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica
marta.calderon@ecci.ucr.ac.cr

Recibido para revisión 25 de Septiembre de 2007, Aceptado 30 de Noviembre de 2007, Versión final 7 de Diciembre de 2007

Resumen—La seguridad del software es una de las mayores preocupaciones de los ingenieros de software. Se deben considerar los requerimientos temprano en el proceso de desarrollo. El objetivo de este artículo es presentar una taxonomía de requerimientos puros de seguridad de software. Tal taxonomía es útil porque sirve como herramienta de educación y puede usarse como lista de comprobación y como guía para la toma temprana de decisiones preventivas. Está generalmente aceptado que la seguridad es una combinación de tres atributos: integridad, disponibilidad y confidencialidad. El no repudio es también una propiedad importante del software seguro. La taxonomía se basa en estos cuatro conceptos y es una jerarquía de dos niveles, en la cual las categorías de primer nivel son requerimientos de integridad, de disponibilidad, de confidencialidad y de no repudio. Usamos esta clasificación primaria porque es fácil para ingenieros de software y usuarios entender los conceptos de integridad, disponibilidad, confidencialidad y no repudio y relacionarlos con los requerimientos funcionales. Para aplicar la taxonomía, se propone un proceso de cuatro pasos: 1) identificar requerimientos funcionales, 2) identificar activos a proteger, 3) identificar amenazas a los activos, y 4) definir requerimientos de seguridad del software. Para mostrar cómo usar la taxonomía, se presenta una aplicación de comercio electrónico.

Palabras Clave—Seguridad, Seguridad del Software, Requerimientos de Seguridad, Integridad, Disponibilidad, Confidencialidad.

Abstract—Software security is a major concern of software engineers. Security requirements must be taken in account early in the software development process. The goal of this paper is to present a taxonomy of software security requirements. Such a taxonomy is useful because it servers as an educational tool, can be used as a check list and as a guide to eliciting software security requirements, can help to creating a software security policy, and can guide to taking early preventive decisions. It is generally accepted that security is the combination of three attributes: integrity, availability, and confidentiality. Non-repudiation is also an important software security property. The taxonomy is based on the four concepts and is a two-level hierarchy, in which the first level categories are integrity requirements, availability requirements, confidentiality requirements and non-repudiation requirements. We use this primary classification because software engineers and users can easily understand the concepts of availability, integrity, confidentiality, and non-repudiation and relate them to functional requirements. To apply the taxonomy, a four-step process is proposed: 1) identify functional requirements, 2) identify assets to be protected, 3) identify threats to the assets, and 4) define software security requirements. To show how to use the taxonomy, an electronic commerce application is used.

Keywords—Security, Software Security, Security Requirements, Integrity, Availability, Confidentiality.

I. Introduction

SECURITY is currently a major concern of software engineers. Much attention has been paid to security flaws introduced during designing or coding phases of the software development life cycle. However, there is need for more security guides to be used in the requirement engineering phase. Although security flaws are rarely introduced in requirement specifications [13], it is highly recommended to consider security requirements early in the software development process [15], in order to prevent introducing vulnerabilities during this development phase. Security requirements can be expressed as security constraints on functional requirements ("shall not … except to") [9], [16]. However, according to [11], most of the time security requirements are defined describing the mechanism used to implement security (e.g., *the system shall provide an authentication mechanism*), but the security issue the requirement is supposed to solve is not clear and security requirements are no related to functional requirements. Therefore, software users cannot identify the threats they, the software application and their assets are exposed to. Hence, it is not possible to establish where (within the software applications) and when (during execution time) security mechanisms must be implemented and activated. Later on in the software development life cycle, security requirements are

translated into other kinds of requirements, such as design, architectural, and implementation requirements [8], without a clear understanding of how implemented security mechanisms are going to affect software system performance and security.

In this paper, we base on two of the principles [16] bases on:

- Security requirements must be understood before they are implemented.
- "Since security is largely concerned with prevention of misuse of system functions, security requirements can most usefully be defined by considering them at the same level as functional requirements, and as constraints upon them" [16, p.3].

The difficulty when creating security requirements is that abnormal behaviors must be anticipated. Identifying abnormal behavior requires developers to think like software hackers and crackers do. However, software engineers do not possess a deep background on security issues [10]. In order to change this situation, software engineers need tools to help them elicit software requirements. These tools must also be intelligible to users, in order to guarantee that engineers and users can work together in the elicitation process and learn about security issues.

Taxonomies – hierarchical systems used to indicate relationships [11] – are useful tools when used as educational tools. In our case, a taxonomy of security requirements can help software engineers and security practitioners concerned about software security requirements. The goal is to help developers more readily identify security problems in the early requirement engineering phase. Moffet *et al.* [16] present some examples of security requirements and mention the need for a taxonomy of requirements. Ahn and Shin [1] mention three kinds of security requirements within the role-based access control approach. There is need for a more comprehensive taxonomy of security requirements [16]. In this paper, we propose a taxonomy of security requirements. The taxonomy is programming language- and development methodology- independent. Taxonomies are useful for the following reasons [10], [18]:

- A taxonomy can be used to help deriving security requirements.
- A taxonomy serves as an educational tool for requirements engineers.
- A taxonomy of security requirements can be used as a checklist to determine whether all kinds of security requirements have been considered.
- A taxonomy can be used as a guide to creating a software security policy.
- For each category in the taxonomy, preventive decisions are taken early in the software development life cycle.

The structure of this paper is as follows. Section 2 describes related work. Section 3 describes the taxonomy proposed. Section 4 describes how the taxonomy can be used in order to elicit security requirements. Section 5 shows an example of how to use the taxonomy. Section 6 presents a discussion about the proposed taxonomy. Finally, Section 7 describes the conclusions of this study.

II. Related Work

Several taxonomies related to software security issues have been published. Reference [13] presents a taxonomy of computer program security flaws which are detected after the system is released and executed in a production environment. References [15] and [19] describe a taxonomy of coding errors impacting security. This taxonomy can be used when developers type or compile code.

Reference [18] presents a two-level hierarchy taxonomy of software security vulnerability causes. The first level corresponds to the software life cycle phase in which vulnerability causes originate and the second level corresponds to the cause. The software life cycle phases considered in this taxonomy are analysis, design, implementation, deployment, and maintenance. For example, according to this taxonomy, in the analysis phase, the three possible causes of software security vulnerabilities are 1) no risk analysis or no security policy, 2) biased risk analysis, and 3) unanticipated risks. However, this classification is not enough to help software engineers elicit software security requirements.

Reference [21] introduces a three-level hierarchy taxonomy of software security flaws. The first level classifies flaws in two categories: inadvertent and intentional. This taxonomy is mostly oriented to flaws introduced during implementation, deployment, and maintenance phases.

Reference [10] presents a taxonomy of software security-related requirements in which four different kinds of security-related requirements are identified: 1) security requirements, 2) security-significant requirements, 3) security-system requirements, and 4) security constraints. Security requirements, also referred to as pure security requirements, refer to defensibility solutions corresponding to integrity, availability, confidentiality, and non-repudiation requirements. Security requirements are classified in four groups: prevention, detection, reaction, and adaptation. They can also be classified as availability, confidentiality, integrity and non-repudiation requirements. Firesmith [10] recognizes that pure security requirements are seldom defined because most of software engineers have received no training in identification, analysis, and specification of this kind of requirements. Security significant requirements refer to non-security requirements, such as functional or quality requirements, having impact on security. Security-system requirements refer to derived security requirements associated with subsystems which implement security functionality, such as access control, encryption/decryption, and antivirus packages. Finally, security constraints are engineering

decisions specified as security requirements, generally implemented for compliance of security policies, standards, or regulation.

The taxonomy in [10] is very useful during the requirement elicitation phase. However, it is still difficult to software engineers to figure out which pure software requirements must be defined using the coarse classification presented. The purpose of our study is to propose a more detailed taxonomy of pure software requirements, in order to provide software engineers with a practical, simple, and familiar security requirement elicitation tool.

III. The Taxonomy

It is generally accepted that security is the combination of three attributes: integrity, availability, and confidentiality [4], [7], [11], [17], [20]. Within the context of this paper, integrity is "the prevention of the unauthorized amendment or deletion of information" [4, p.6], availability is "the prevention of the unauthorized withholding of information" [4, p.6], and confidentiality means "the prevention of the unauthorized disclosure of information" [4, p.6]. This security definition is the basis of the approach used in this study to define a taxonomy of software security requirements. Following this definition, security requirements are divided into three main categories:

1. *Integrity constraints:* constraints specifying how to preserve system integrity.
2. *Availability constraints:* constraints specifying how to preserve system availability.
3. *Confidentiality constraints:* constraints specifying how to preserve system confidentiality.

We decided to use this primary classification plus non-repudiation requirements because software engineers and users can easily understand the concepts of availability, integrity, confidentiality and non-repudiation, and relate the effects of these non-functional software properties to the system functional requirements. Because the taxonomy is intended to be used in the early requirements elicitation phase of the software development life cycle, the purpose is to identify security requirements using plain language easily understandable for software engineers and users. No decisions about which and how security mechanisms will be implemented are included in security requirements.

Conflicts among security requirements are common. Software engineers and especially users have to decide the emphasis of each of the attributes (integrity, availability, or confidentiality), based on concerns and interests of the stakeholders and on the nature of the software system domain.

The taxonomy presented in this paper is a two-level hierarchy, as shown in Table 1. The second-level categories correspond to more specific security issues. The taxonomy was built combining personal experience and input from several sources: a literature survey of other existing security issues taxonomies [3], [10], [11], [13], [18], [19], [21], and of papers describing security requirements, constraints, policies and attacks [2], [12], [14]. The rest of this section describes each of the second-level categories.

Table 1. The Taxonomy of Security Requirements

First level categories	Second level categories
Non-repudiation requirements	Non-repudiation
Integrity requirements	Modification
	Deletion
	Datum validation
	Exception handling
	Prerequisite
	Separation of duties
	Temporal
Availability requirements	Response time
	Expiration
	Resource allocation
Confidentiality requirements	Encryption
	Authentication
	Aggregation
	Attribution
	Consent and notification
	Cardinality
	Traces

A. Non-repudiation requirements

Non-repudiation refers to the situation in which a person cannot claim to have not performed some action. Transactions requiring non-repudiation shall be identified. This kind of constraints is especially important in e-commerce systems.

Example: The system shall not allow a purchase transaction before the system knows the identity of the customer.

B. Integrity requirements

Security requirements are required to restrict actions threatening integrity. Second-level constraint subcategories are discussed below.

1) Modification requirements: Modification of restricted or sensitive data must be constrained. Circumstances under which modifications are allowed must be defined. Modification logs required must be identified.

Example: The system shall not allow customers change sale price.

2) Deletion requirements: Deletion constraints identify the circumstances under which information can be deleted. Deletion logs required must be identified. When deletion

constraints are defined, persistence constraints must be considered. Persistence refers to the set of rules on how old records must be kept accessible in data bases. Legislation and regulation define persistence constraints for some kinds of data such as medical or financial records. Persistence constraints prevent from deleting information until the appropriate time is expired.

Example: The system shall not allow users delete historic accounting records generated during the last five years.

3) Datum validation requirements: Input data requiring validation must be identified. Valid values, ranges, types, and sizes must be clearly defined to constraint input.

Example: The system shall not allow users input strings longer than the buffers used to stored input data.

4) Exception handling requirements: Exception handling is important to avoid integrity violations. Possible exceptions and their corresponding handling approaches must be identified.

Example: The system shall inform the user when she inputs the value zero in the input box containing the number of months used to calculate an average salary.

Example: The system shall inform the user when the system does not find the payroll file and shall assure that the software application does not fall abruptly.

5) Prerequisite requirements: If following an order in which actions take place in a system is necessary in order to maintain integrity, this order must be forced through prerequisite constraints. For example, a user request must be authorized before being processed.

Example: The system shall not allow users to withdraw money from their account before the system checks funds availability.

6) Separation of duty requirements: In order to establish internal controls and to prevent frauds, separation of duty constraints must be defined. Conflicting roles or tasks must be identified.

Example: The system shall not allow a Purchase Department member to use account payable system options.

7) Temporal requirements: Under some circumstances, some user actions may be allowed only during certain time periods, such as office hours or before a due date or time is reached. This kind of constraints can be applied to stock exchange and medical information systems.

Example: The system shall not allow stock exchange transactions outside normal stock exchange hours.

C. Availability requirements

In order to guarantee continued service, availability constraints must be considered. Requirement subcategories are detailed below.

1) Response time requirements: Constraints on response time may be used to define availability requirements. For example, the percent of requests which has to be serviced within certain period is defined through a response time constraint.

Example: The system shall provide student information within 1 hour for 99% of requests.

2) Expiration requirements: This kind of constraints restricts the amount of time a connection takes to expire, that is, to time-out, because of user inactivity. A large amount of connection requests by an illegitimate user may prevent the system from processing of legitimate user requests, affecting availability. In order to reduce the effect of such a kind of attacks, the amount of time to time-out must be low enough. This kind of constraints can also be considered a confidentiality constraint useful when users forget to log-out and leave their working places.

Example: The system shall reduce the time connection requests take to time-out to 1 minute when the number of connection requests exceeds 10,000 per hour.

3) Resource allocation requirements: Resources such as memory, disk space and CPU time allocated to a user must be restricted in order to avoid asset deprivation and unavailability. This is especially important when anonymous users are allowed. Amounts of free memory and disk space required to handle user requests must be defined.

Example: The system shall not assign a single user more than 100 MB in hard disk.

Example: The system shall not assign more than 50% of all work memory available for user requests.

D. Confidentiality requirements

Confidentiality requirement subcategories are discussed below.

1) Encryption requirements: Sensitive information which must be encrypted before being stored or transmitted must be identified.

Example: The system shall not allow users to transmit credit card numbers using an easily understandable format.

2) Authentication requirements: According to [11, p.116], authentication is "concerned with ensuring that users are who they say they are, and remain so during the 'session'." Working schemes requiring sessions and transaction types requiring strong or very strong authentication approaches must be identified.

Example: The system shall not allow users withdraw money from their personal accounts before they re-authenticate.

3) Aggregation requirements: Aggregates of data may provide valuable information. Access to aggregated information derives from having access to a large number of records. Aggregation constraints must be defined in order to prevent users from having access to aggregated information which must not be disclosed. Examples of aggregated information are daily sales reports and monthly customer purchase reports.

Example: The system shall not allow tellers to access daily sales reports before they execute the end day drawer process (in a Point of Sales system).

4) Attribution requirements: Access logs must be defined. Access logs must contain details such as subject name, date and time, in order to identify unauthorized disclosure of information. Logs of deletions from access logs must be defined depending on how sensitive the information is.

Example: The system shall not allow users delete records from the account access log except to system administrator.

Example: The system shall record user identification, date, and time each time a user prints a customer list.

5) Consent and notification requirements: This concept is closely related to privacy. Depending on the nature of a software system, consent and notification constraints must be defined. Datum owners have to be notified of the people with right to access their data or asking to access it. This kind of constraint is important in medical information systems. Legislation regulating this kind of constraints must be considered.

Example: The system shall not allow medical center physicians to have access to medical records of a patient unless the patient has approved the access.

6) Cardinality requirements: The number of simultaneous connections or sessions a user can hold must be defined. Besides being used to preserve confidentiality, this kind of constraints is also important to avoid system overload and possible system unavailability.

Example: The system shall not allow more than two simultaneous connections to a user.

7) Requirements on traces: Traces on previously accessed information may be required depending on the nature of the system. Mutual exclusion constraints may be required. For example, if a software system offers two mutually exclusive options, once a user has chosen one option, she cannot choose the other one. An interesting example of this constraint category is described in [6]. In this example, organizations are classified in conflict of interest classes. Organizations belonging to one conflict of interest class are competitors among them. People are allowed to access information about only one organization per conflict of interest class. In order to achieve this goal, it is necessary to keep trace of previous access to the information.

Example: The system shall not provide organization information access to any person who previously accessed information about another organization within the same conflict of interest class [6].

IV. How to Use the Taxonomy

The taxonomy of software security in conjunction with functional requirement descriptions can be used as a guide to eliciting software security requirements. The elicitation process is a four-step process, as shown in Fig. 1. Steps are:

1) Identify functional requirements: Security requirements make sense within a functional requirement context. Therefore, functional requirements need to be identified. UML use case diagrams and textual descriptions can be used to specify functional requirements.

2) Identify assets to be protected: Based on functional requirements, software engineers can identify critical assets which have to be protected in order to guarantee integrity, confidentiality, and availability. Both information assets and physical assets must be identified [11]. For example, in an electronic commerce system, one of the information assets to be protected is the product catalog. Determining how to identify assets is out of the scope of this study. Haley *et al.* [11], Moffett *et al.* [16], and Myagmar *et al.* [17] provide valuable information about how to identify assets.

3) Identify threats to the assets: According to [11, p. 112], "a threat is the potential for abuse of an asset that will cause *harm* in the context of the problem." Threats can be described relating *assets*, *actions*, and *harm* [11]. Harm can be caused to the organization developing the software or to its customers, providers and other stakeholders. Security requirements aim to protect assets from threats [4]. Therefore, identifying threats is very important in order to help locate where security requirements have to be defined. When assets to be protected have been identified, software engineers and users can identify threats which may compromise asset integrity, availability, or confidentiality. For example, in the electronic commerce system, the product catalog (*asset*) can be modified or deleted by an illegal user (*actions*), resulting in revenue loss and reputation damage (*harm*). Determining how to identify threats is out of the scope of this study. Haley *et al.* [11], Moffett *et al.* [16], and Myagmar *et al.* [17] emphasize on the importance of threats are a basis for security requirements and provide valuable information about how to identify threats.

4) Define software security requirements: Take each threat identified in step 3 and check each subcategory in the taxonomy to define requirements. Software engineers and users can ask a question such as, "Is there a way the absence of X requirements can contribute to execute action A?" In this case, X represents a sub-category in the taxonomy and A represents the action in a threat description. Following the threat example from step 3 (product catalog modification) and starting from the *Non-repudiation* category, *Non-repudiation* constraint absence does not contribute to the catalog modification threat, whereas *Modification* constraint absence – from the *Integrity* category –does. Software engineers and users must determine under which circumstances catalog modification is not allowed. For example, software engineers and users determine that members of the Inventory Department are allowed to modify the product catalog but customers and employees of any other departments cannot modify it. Then, one or more security requirements are

defined. As an example, the following security constraint is defined:

"The system shall not allow users change the product catalog except to Inventory Department members."

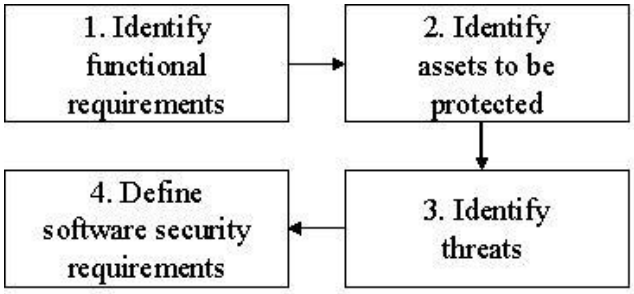

Figure 1. Security Requirement Elicitation Process

Notice that not all taxonomy subcategories are applicable to every software application. In fact, applicability depends on the application nature. It is also possible to find out that there are conflicting software security requirements. There can also be conflicts among stakeholders, due to differences in interests. In such a case, users, clients, software engineers and other stakeholders have to reach an agreement about which requirements are more important and which have to be totally or partially sacrificed. However, this is out of the scope of this study.

It is important to emphasize that software security requirements can help to defining preventive decisions but do not show any details about security mechanism implementation. Security mechanisms will be defined later in the software development life cycle.

The list of security requirements obtained can be used later in system validation and verification processes, especially when testing security. Sometimes, different requirements related to the same asset can be combined, as far as they are not conflicting, in order to reduce the number of requirements. For example, constraint "*the system shall not allow users change the product catalog except to Inventory Department members during office hours*" is the combination of a *Modification* constraint and a *Temporal* constraint.

V. Example of Taxonomy Use

We will illustrate how to use the proposed taxonomy using a Web-based hotel reservation system. We will follow the four-step process proposed to elicit software requirements

A. Identify Functional Requirements

In this example we will use the Unified Modeling Language (UML) [5] to represent functional requirements. Fig. 2 shows the corresponding UML use case diagram. As shown in Fig. 2, the hotel reservation system consists of ten use cases. Textual descriptions for *Change Reservation* and *Guarantee Reservation* use cases are shown below. We develop this example using only these two use cases, but the whole process must be executed to cover the whole system functionality.

Use case name: Change Reservation
Summary: A customer changes one or more details into an existing room reservation, via Web.
Dependency: Extend Guarantee Reservation Abstract Use Case.
Actor: Principal Actor: The customer.
Precondition: The reservation had been previously made and the system is idle.
Description:

1. The customer enters the reservation code or the customer's name.
2. The system displays all the information relative to the reservation.
3. The customer enters the information to be changed (check-in date, check-out date, numbers of persons, and quantity and type of rooms).
4. The system checks room availability according to data entered by the customer.
5. The system changes reservation, displays a successful message and the reservation code.
6. If the customer wants to guarantee the reservation or change the credit card number: Extend Guarantee Reservation Abstract Use Case.

Alternatives:

- If the hotel does not have rooms available, the system prompts to conserve the original reservation and display an apology message explaining the exact reason. Customer may provide other data to try again. If the customer doesn't want to conserve the original reservation, the system prompts to cancel the reservation and displays an apology message.
- If the reservation could not be guaranteed because the credit card number is invalid, the system asks the customer whether she provides another credit card number or prefers to guarantee the reservation later.

Postcondition: Reservation is changed.

Use case name: Guarantee Reservation
Summary: The system guarantees the reservation.
Dependency: None.
Actor: Principal Actor: The customer.
Secondary Actor: The external credit card system.
Precondition: The customer has previously made a reservation.
Description:

1. The customer enters credit card data (credit card number, expiration date of the card, and credit card owner name).
2. The external system validates the credit card number provided (existence, non stolen, non expired).
3. The system associates the credit card number with the reservation.

Alternatives:

- If the external system does not validate the credit card number, the system returns a message to indicate it is not a valid credit card.

Postcondition: The reservation was guaranteed.

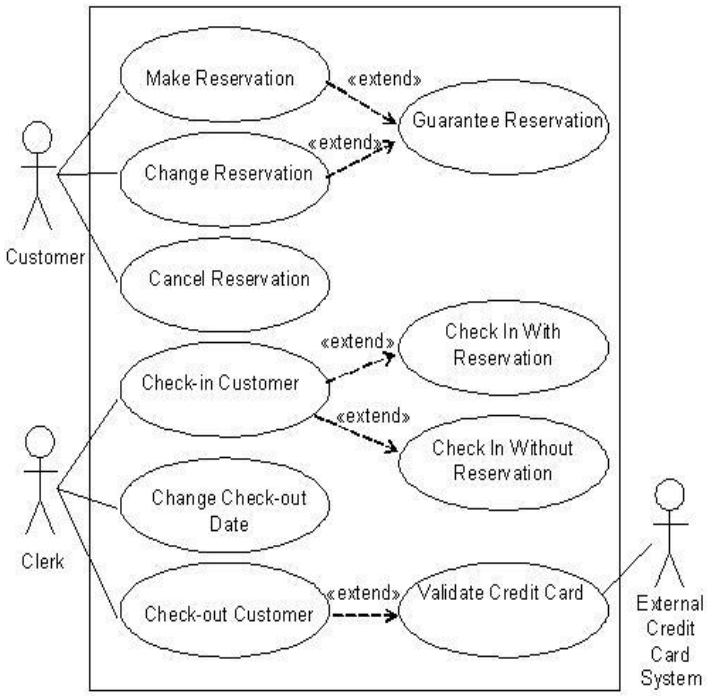

Figure 2. Use Case Diagram for a Web-based Hotel Reservation System

B. Identify Assets to Be Protected

From the textual descriptions corresponding to the two use cases, software engineers and users derive that the system reads or stores information about reservations and credit cards. These are the assets to protect.

C. Identify Threats to the Assets

Software engineers derive a list of threats associated with reservations and credit cards. For example, in this case some of the threats are:

a) Reservation information may be modified by an illegal user. This situation can damage reputation of the hotel.

b) Reservation information may be changed by a legal user who later denies she made the change, provoking a revenue loss and a reputation damage.

c) Credit card information may be accessed and disclosed by an illegal user, leading to possible lawsuits or reputation damage to the hotel and financial loss to the credit card owner.

Note that there are more possible threats associated with the identified assets. However, in this example we only consider threats associated with the functionality under analysis, that is, the *Change Reservation* and *Guarantee Reservation* use cases.

D. Define Software Security Requirements

For each threat identified in the previous step, follow these steps: 1) take each of the taxonomy categories, 2) determine whether the absence of requirements of the category contributes to the action associated with the threat, and 3) define corresponding security requirements. Table 2 shows software security requirements related to threats listed above.

VI. Discussion

The software taxonomy proposed in this study has some limitations. It is a complement to the security-related requirement taxonomy presented by Firesmith in [10]. The taxonomy only focuses in one of the four categories specified by Firesmith: the pure security requirements. The focus is set on software security requirements which can be elicited during the requirement engineering process, independently of the security mechanisms established later in the software life cycle. Eliciting security requirements takes time, not always available to software engineers. Learning to identify threats and define corresponding security requirements also takes time. Additionally, using the taxonomy does not warranty that the resulting security requirements will be verifiable.

Despite these limitations, using the proposed taxonomy offers several advantages to software engineers. It can help them to identify the threats software applications and their assets are exposed to. Security mechanisms can be designed to solve specific threats and security issues. Well written security requirements, that is, unambiguous and feasible, are verifiable. This will not ensure that software systems are completely secure, but it is possible to assert that software assets are secure to a certain degree. Software engineers can create test cases for each security requirement. They can also trace implemented security mechanisms to security requirements.

It is also important to emphasize on the necessity of defining security requirements which do not constrain the options security engineers can choose when taking architectural decisions.

Table 2. SECURITY REQUIREMENTS CORRESPONDING TO CHANGE RESERVATION AND GUARANTEE RESERVATION USE CASES

First level categories	Second level categories	Security requirements
Non-repudiation requirements	Non-repudiation	The system shall not allow users change reservations before the system provides a non-repudiation mechanism (Threat b).
Integrity requirements	Modification	The system shall not allow users change other users' reservations (Threat a).
	Deletion	
	Datum validation	
	Exception handling	
	Prerequisite	
	Separation of duties	
	Temporal	
Availability requirements	Response time	
	Expiration	
	Resource allocation	
Confidentiality requirements	Encryption	The system shall not allow credit card information storage/transmission using an easily understandable format (Threat c).
	Authentication	The system shall not allow users change a reservation before users' identity is known and remains the same when the reservation is changed (Threats a and b).
	Aggregation	
	Attribution	
	Consent and notification	
	Cardinality	
	Traces	

VII. CONCLUSIONS

In this paper a general purpose, two-level hierarchy taxonomy of software security requirements is described. The first level categories are integrity requirements, availability requirements, confidentiality requirements, and non-repudiation requirements. Each first level category is subdivided in more detailed requirements. The taxonomy is useful during the requirement engineering phase and must be adapted to the particular security requirements of a software system.

To apply the taxonomy, a four-step process is proposed: 1) identify functional requirements, 2) identify assets to be protected, 3) identify threats to the assets, and 4) define software security requirements. When conflicts among security requirements arise, software engineers and users have to decide which security requirements prevail and which are partially or totally discarded. If the resulting security constraint list is very long, requirements related to the same asset can be combined into one.

Several research issues remain. First, it is important to understand conflicts arising among different kinds of security requirements and how they can be solved in order to generate a consistent security constraint set. Also, the taxonomy may be specialized to specific domains, such as electronic commerce, banking, or medical records.

REFERENCES

[1] G. J. Ahn, and M. E. Shin, "Role-based authorization constraints specification using object constraint language," in *Proceedings of the Tenth IEEE International Workshop on Enabling Technologies: Infraestructure for Collaborative Enterprises*. (WETICE 2001), (Cambridge, MA, June 20-22), pp. 157-162.

[2] R. J. Anderson, *Security in Clinical Information Systems*. Technical Report, British Medical Association 4, 1996.

[3] T. Aslam, I. Krsul , and E. Spafford, "Use of a taxonomy of security faults," in *Proceedings of 19th NIST-NCSC National Information Systems Security Conference*, 1996, pp. 551-560

[4] A. Avižienis, J. C. Laprie, J.C., and B. Randell, *Fundamental Concepts of Dependability*, UCLA CSD Report no. 010028, University of California at Los Angeles, 2000.

[5] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Reading, MA: Addison Wesley. 1999.

[6] D. F. C. Brewer, and M. J. Nash "The chinese wall security policy," in *Proceedings of the IEEE Symposium on Research in Security and Privacy* (Oakland, CA, May 1989), pp. 206-214

[7] R. Crook, D. Ince., L. Lin, and B. Nuseibeh, "Security requirements engineering: when anti-requirements hit the fan," in *Proceedings of Joint International Requirements Engineering Conference* (RE'2002), IEEE Computer Society, pp. 203-205.

[8] D. G. Firesmith., *Common Concepts Underlying Safety, Security, and Survivability Engineering,* Technical Note CMU/SEI-2003-TN-033, Software Engineering Institute, Pittsburgh, Pennsylvania, December, 2003.

[9] D. G. Firesmith, "Specifying Reusable Security Requirements," in *Journal of Object Technology*, Vol 3, No. 1 (January-February, 2004), pp. 61-75.

[10] D. G. Firesmith, "A Taxonomy of Security-Related Requirements," in *Proceedings of the 27th international conference on Software engineering*. New York, USA: ACM Press, pp. 720 – 721, 2005

[11] C. B. Haley, R. C. Laney, C. Robin, and B. Nuseibeh, "Deriving security requirements from crosscutting threat descriptions," in *Proceedings of the 3rd International Conference on Aspect Oriented Software Development* (AOSD'04), Lancaster, UK, 22-26 March 2004, New York, NY: ACM Press, pp. 112-121.

[12] L. Labuschagne, "A framework for electronic commerce security," in *Proceedings of the IFIP TC11 Fifteenth Annual Working Conference on Information Security for Global Information Infrastructures*. Edited by S. Qing and J. H. P. Eloff. Fifteenth Annual Working Conference on Information Security (Beijing, China, Aug. 22-24, 2000), Deventer, The Netherlands: Kluwer Academic Publishers, pp. 441-450.

[13] C. Landwehr, A. Bull, J. McDermott, and W. Choi, "A taxonomy of computer program security flaws, with examples." *ACM Computing Surveys*, Vol. 26, No. 3 , pp. 211–255, 1994.

[14] R. C. Linger, and A. P Moore, *Foundations for Survivable System Development: Service Traces, Intrusion Traces, and Evaluation Models*, Technical Report CMU/SEI-2001-TR-029 ESC-TR-2001-029, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, United States, 2001.

[15] G. McGraw, *Software Security: Building Security In*. United States: Addison Wesley, 2006.
[16] J. D, Moffett, C. B. Haley, C.B., R. C. Laney, and B. Nuseibeh, *Core Security Requirements Artefacts*, Technical Report No 2004/23, Department of Computing , The Open University, Milton Keynes, United Kingdom, June, 2004.
[17] S. Myagmar, A. J. Lee, A.J., and W. Yurcik, "Threat modeling as a basis for security requirements," in *Proceedings of the IEEE Symposium on Requirements Engineering for Information Security* (SREIS), 2005.
[18] F. Piessens, "A taxonomy of causes of software vulnerabilities in Internet software," in *Supplementary Proceedings of the 13th International Symposium on Software Reliability Engineering, Annapolis*, (Maryland, U.S.A., November 2002), IEEE Computer Society, IEEE Press, pp. 47-52
[19] K. Tsipenyuk, B. Chess, B., and G. McGraw, "Seven pernicious kingdoms: a taxonomy of software security errors," in IEEE *Security & Privacy Magazine*, Vol. 3, No. 6 (Nov.-Dec., 2005), pp. 81-84, 2005.
[20] J. Viega, and G. McGraw, G. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison Wesley, 2001.
[21] S. Weber, P. A. Karger, and A. Paradkar, "A software flaw taxonomy: aiming tools at security," in *Proceedings of the 2005 Workshop on Software Engineering for Secure systems|-building Trustworthy Applications* (SESS'05). New York, NY, USA: ACM Press, pp. 1-7.

Marta Calderón obtained a Masters of Science in software engineering at Texas Tech University, Lubbock, Texas in 2003 and a Masters in business administration at INCAE Business School, Alajuela, Costa Rica in 1990.

She is an Associate Professor at the Department of Computer Science and Informatics at the Universidad de Costa Rica, San Pedro, Costa Rica. Her research interests are software security, software validation and verification, and human-computer interaction.