A Catalogue of Reusable Security Concerns: Focus on Privacy Threats

(Research-in-Progress Paper)

Luís Gonçalves, Alberto Rodrigues da Silva
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

Abstract—With the increasing number of cyber-attacks, organizations are giving more importance to secure their systems. Security shall be considered from the very beginning, including requirement engineering processes to prevent common vulnerabilities as well to avoid re-work costs. However, there is not yet an appropriate approach to specify such security requirements in a rigorous and systematic way. Such approach would allow defining and specifying security-specific concepts like vulnerabilities, threats or attacks. In addition, it would support the catalogue structure with reusable security requirements. The strategy we follow to address this challenge involves the following aspects: First, we decide to use the recent RSLingo RSL language as the underline for a rigorous requirements specification language. Second, we extend this language by including security-specific concepts with a comprehensive requirements classification schema involving the following aspects: solution versus problem, abstract versus concrete and positive versus negative requirements. Third, we apply this extended language with an illustrative sample of security requirements and other concepts, which can be easily reused and extended by the community. The current version of this catalogue aggregates 20 packages, one of which is the privacy concerns package that is the focus of this paper and includes currently 51 security requirements, 27 vulnerabilities, and 21 threats.

Index Terms— Requirements specification; Cyber-security; Catalogue of security requirements

1 Introduction

System security has been achieved by using systematic engineering approaches. Several methods and techniques have been developed to protect data, programs and networks from attacks or other infringements through mechanisms such as access controls and firewalls [1]. However, the majority of organizations tend to see the importance of security only after actually suffering damage from security breaches. Even when security is considered important, too often it is retrofitted into the application only at the end of the construction phase and leads to many vulnerabilities and negative impacts [2,3].

In the past years there has been an increasing number of proposals with approaches to help and improve the threat elicitation and modelling of such situations [4,5] and that also included models and frameworks to support security specifications [6,7,8]. But still there was not a consensus on which method provides the best results. Moreover, some of these approaches do not consider the entire problem; they only present a partial perspective of the whole picture. For example, Sindre et al. [4] extend the traditional use case concept to consider misuse cases, or McDermott et al. [9] propose a specification of interactions that should not happen between a system and its actors called abuse cases. However, none of them related these use case specifications with the concepts of threats or vulnerabilities. On the other hand, for example FIRST [10] offers a set of practices and tools to define and characterize vulnerabilities but it does not explain how to relate them with threats or with requirements.

In this research we use and extend the RSLingo RSL language [8,11,12] to include security-specific concerns. RSL is a controlled natural language designed to help the production of requirements specifications in a systematic and rigorous way. It allows specifying requirements through a rich set of constructs logically arranged into views according to multiple concerns that exist at both business and system abstraction levels. RSL is the base for our work, but, since it did not support security-specific concerns (such as vulnerabilities, threats or attacks), we explain how to extend it for this purpose. We follow a pragmatic approach to implement such extension, namely by directly update the RSL language and the Excel template, both publicly available [11]. For the sake of simplicity, in the context of this paper, we named "RSL4Cybersecurity" as this RSL language extended version. Finally, we also discuss the definition and structure of a catalogue of security-specific concerns that may support the requirements specification with well-understood solutions to recurring security problems and that can be reusable and extended. In the context of the paper, we named "Catalogue4CyberSecurity" this set of well-defined and classified specifications.

This paper is structured in 8 sections. Section 2 introduces the RSL language that was adopted to support the specification of security-specific concerns. Section 3 dis-

cusses the RSL extension proposed as a set of additional requirement properties as well as new security concepts. Section 4 proposes the structure of a catalogue with reusable security concerns. Sections 5 and 6 discuss a privacy package of the Catalogue4CyberSecurity and its application to a web app example. Section 7 introduces and discusses the related work that most influenced our research. Lastly, Section 8 presents the conclusion and identifies open issues for future work.

2 RSLingo RSL

RSLingo has been a long-term research initiative in the requirement engineering (RE) area that recognizes that natural languages, although being the most common and preferred form of representation used within requirements documents, are prone to produce ambiguous and inconsistent documents that are hard to automatically validate or transform [7]. Recently, we propose a broader and consistent language, called "RSLingo RSL" (or just "RSL" for brevity), based on the design of former languages [8]. RSL is a controlled natural language designed to help the production of requirements specifications in a systematic, rigorous and consistent way [8,12]. RSL includes a rich set of constructs (e.g. stakeholders, actors, use case, user stories, systems goals, quality requirements) logically arranged into views according to RE-specific concerns that exist at both business and system levels. These constructs are also defined as linguistic patterns and represented textually by mandatory and optional fragments [8].

Fig. 1 shows a partial view of the RSL metamodel, that defines a hierarchy of requirement types, such as business and system goal, use case etc. A requirement can aggregate other requirements (through the "isPart" relationship) and may establish different types of relationships with other requirements (through the "Requirement Relation" relationship); this relationship can be further classified (by the "requirements relationship kind" enumeration). RSL has been implemented with the Xtext framework [13] and, consequently, its specifications are rigorous and can be automatically validated and transformed into other representations and formats. In addition, a lightweight tool support is also provided based on an RSL Excel template publicly available at ResearchGate [11]. This RSL Excel Template has been extensively used as a starting point for our research before implementing the changes to the RSL language in the Xtext framework [13]. In spite of the extensive set of elements and features, RSL still lacks the support for security requirements specification. For instance, RSL allows specifying security requirements as quality requirements but there are no concerns to support the specification of threats, vulnerabilities or the involved relationships.

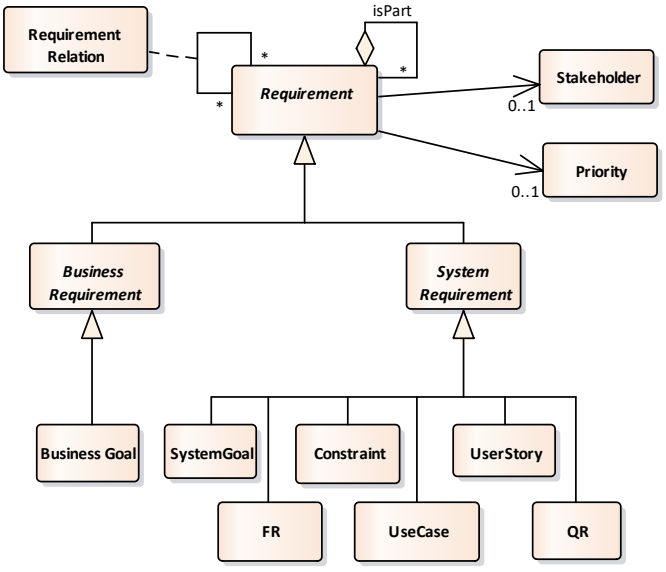

Fig. 1. RSL partial metamodel: the hierarchy of requirements.

3 RSL4Cybersecurity language

As briefly introduced in Section 2 there are many concepts and terms that emerge from the literature when we want to specify and analyze system's security concerns. Some of these concepts are: vulnerabilities, attacks, threats, risks, treatments measures, requirements, goals, misuse cases and abuse cases. In this section we discuss how to properly define and relate these common concepts. To support our discussion, we introduce a set of sentences that exemplify some of these concerns: sentences S1 to S6 in Table 1.

Based on these sentences we discuss the need for extending the core of the RSL language for a better classification of requirements (Section 3.1) and then for specifying other security-specific concerns (Section 3.2).

Table 1. Example of sentences with security concerns.

S1	System shall guarantee that communications between the client and the server are kept private by some encryption technique.
S2	System shall guarantee that communications between the client and the server are kept private by using TLS 1.3 protocol.
S3	A vulnerability in the login form allows the attack to inject code into the database.
S4	The sanitization of the data received from untrusted source prevents injection of code.
S5	An attacker can use vulnerabilities of MongoDB to access and modify a database.
S6	Dave inserts an infected USB pen into a company computer and infects the network of the company with malware.

3.1 Extended Classification of Requirements

RSL language provides a large set of concepts that allow specifying different types of requirements, such as business and system goals, use cases, quality requirements, constraints or user stories [8,12]. In addition, each type of requirement has its own specific classification schema. For instance, system goals can be further classified as security, usability etc. However, from the analysis of the related work (see Section 7) and also from our preliminary attempts to specify reusable requirements, we verified the need for some extension schema to better classify these requirements. RSL4Cybersecurity extends the original RSL language with security concerns.

In this way, we address the needs of system designers to specify security requirements aligned with the goals of their system. This language extension starts by redefining the Requirement concept. As shown is Table 2, we introduce three new properties to specify a requirement: *IsPositive*, *isAbstract* and *isProblem*. By default, and if not explicitly stated, a requirement is classified as a *positive, abstract and problem-oriented* requirement.

The *isPositive* property allows to state if the requirement is positive or negative. If the requirement is positive then it means it shall be supported or provided by the system, while a negative requirement shall be avoided or at least mitigated. A negative requirement is used mainly in security and safety contexts when we want to specify unwanted situations, commonly referred as *misuse cases* or *abuse cases*. An example of a negative requirement is the sentence S6 that briefly defines a negative use case (or misuse case).

The *isAbstract* property allows to state if the requirement is abstract or concrete. An abstract requirement means it is specified in a somehow general and imprecise way, while a concrete requirement that it is specified in a more specific and clear way. Of course, this classification is itself subjective, hard to apply and directly depends of its context and the background of the people involved. An example of an abstract and concrete requirement is, respectively, S1 and S2: S1 ["…data… are kept private"] refers a confidential-related goal in an abstract way; while S2 ["data… kept private by using TLS 1.3 protocol"] extends the previous requirements with some more specific information.

The *isProblem* property allows identifying if the requirement is stated as a problem or as a solution (for some problem). In general, a requirement is stated as a problem and consequently focusses on the *what* of the system; that is the common situation, for example of *use case* based specifications. However, in other situations, we want requirements that explicitly express the *solution* (i.e., the *how*) for an identified problem. We may also adopt the expression "requirement patterns" for the solution-oriented requirements. An example of a solution requirement is S4 that recommends the sanitization of the data received from untrusted sources to avoid injection-based attacks. Table 2 summarizes the classification of requirements according to these additional properties, and Table 3 classifies the sentences introduced above in Table 1, according to this schema.

Table 2. Summary of the additional requirement properties

	isPositive	**isAbstract**	**isProblem**
Yes	Positive requirement	Requirement defined in an abstract way	Requirement defined as a problem
No	Negative requirement, e.g. misuse case or abuse case	Requirement defined in a more concrete way	Requirement defined as a solution

Table 3. Classification of the security-specific concerns

Sentence	**Requirement type**	**isPositve**	**isAbstract**	**isProblem**
S1	System Goal	Yes	Yes	Yes
S2	System Goal	Yes	No	Yes
S3	Vulnerability	-	-	-
S4	System Goal	Yes	No	No
S5	Threat	-	-	-
S6	Misuse Case	No	No	Yes

3.2 Security-Specific Extension

In the scope of security specification there are some concepts that are relevant to state but shall not be considered or classified as requirements because they do not represent needs or wishes from the stakeholders. The sentences S3 and S5 are examples of such information that refer other concepts, namely vulnerabilities (S3), threats or risks (S5). As a consequence, new questions appear when we introduce these new concepts, for instance: what properties shall be added for defining them?, what classification schema shall be adopted?, or how we shall relate requirements with threats and vulnerabilities?

Fig.2 illustrates our proposal to answer such questions: it illustrates the partial meta-model of the proposed security-specific extension of RSL. The key concepts are threats and vulnerabilities. *Threats* and *risks* are synonymous in the sense that they identify some event that can happen, and usually, with a negative consequence. Because in this security domain the term "threat" is more commonly used than "risk", we adopted that term. A threat can be associated to several well-known vulnerabilities of some asset (e.g., the entire system, the supported operating system, databases). In addition, for each threat, we can define multiple treatment measures. Such treatment measures are classified by a treatment strategy (e.g. avoid, mitigate, transfer or accept) and can be assigned to a security-specific requirement, preferentially a "solution-oriented" requirement previously defined.

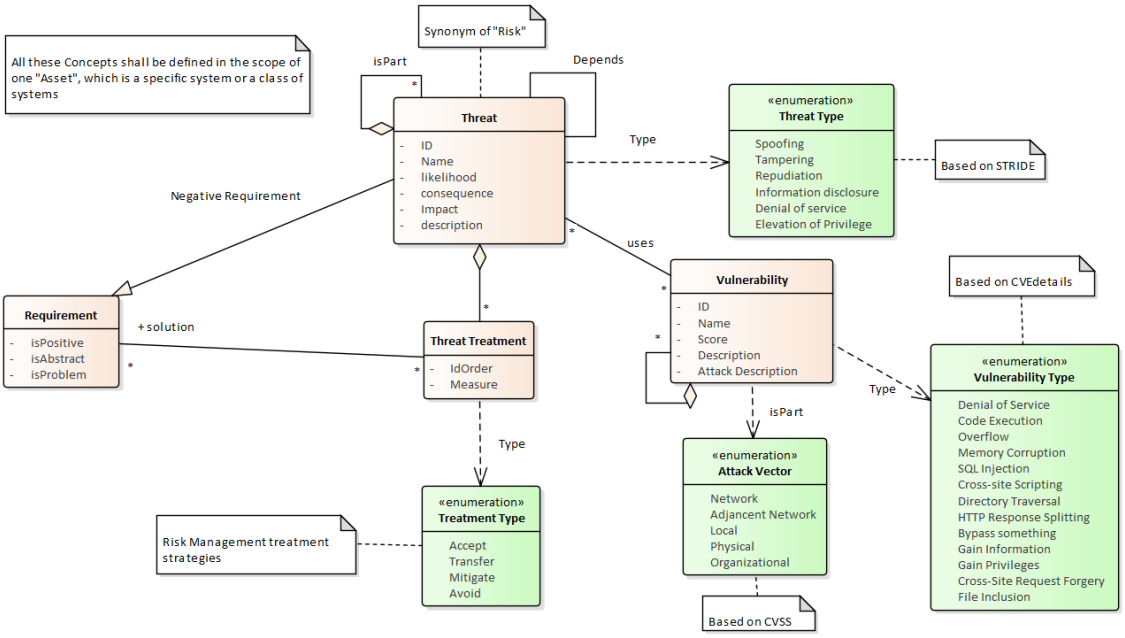

Fig. 2. RSL4CyberSecurity metamodel (partial view).

3.3 Threat Classification

A threat is the possibility of harming an asset of our system (or the system as a whole). We adopt the STRIDE threat taxonomy [14] to identify security threat types. STRIDE is an acronym for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege. These threats are the negation of security properties, respectively: confidentiality, integrity, availability, authentication, authorization and non-repudiation.

- *Spoofing* (negates *Authentication*): Illegal access and use of another user's authentication information, such as username and password; an example could be a person outside a company using credentials of the company employee to access the system.
- *Tampering* (negates *Integrity*): Involves the malicious modification of data; an example includes unauthorized changes made to persistent data, such as that held in a database.
- *Repudiation* (negates *Non-repudiation*): When users deny performing an action without other parties having any way to prove otherwise; for example, a user performs an illegal operation in a system that lacks the ability to trace the prohibited operations.
- *Information Disclosure* (negates *Confidentiality*): The exposure of information to individuals who are not supposed to have access to it; for example, the ability of users to read a file that they were not granted access to, or the ability of an intruder to read data in transit between two computers.
- *Denial of Service (DoS)* (negates *Availability*): Attacks that deny service to valid users; for example, by making a web server temporarily unavailable or unusable.
- *Elevation of Privilege* (negates *Authorization*): When users get unauthorized access to some resource or feature; for example, when an unprivileged user gains privileged access and thereby has sufficient access to compromise or destroy the system.

3.4 Threat Treatment

A threat severity shall be defined in order to prioritize and treat it accordingly, and for that purpose some threat impact model can be used. The proposed threat impact model considers as the input the likelihood and the consequence of the threat and calculates the threat impact level with the mean value of the two previous parameters, for example in a 0-10 range, from 0 (no impact) to 10 (maximum impact). With the severity specified, it is possible to define how to treat each threat according to the following strategies:

- *Accept*, the threat is so low and so costly to mitigate that it is worth just accepting it;
- *Transfer*, transfer the threat to some other entity via insurance, contracts etc;
- *Mitigate*, reduce the threat impact level with countermeasures, such as requiring a long password, an encrypted communication;
- *Avoid*, remove the system component or feature associated with the threat because the threat impact is too high and the organization cannot cope with it.

After being identified which type of treatment can best address the threat, the user can also specify the concrete measure or solution for that problem. For example, a system goal can be assigned as a convenient solution to treat such threat defined as a security pattern.

3.5 Vulnerabilities

Another concept directly connected to a threat is the Vulnerability, since a threat exists if there is at least one vulnerability to exploit. Vulnerability is a flaw in the system that opens it to an attack or exploit which will lead to damage or loss of property. With that in mind, the Vulnerability concept introduced in RSL4Cybersecurity extension is based on the Common Vulnerability Scoring System (CVSS) [10]. The objective of CSVV is to gather the characteristics of vulnerabilities and give them a score, reflecting its severity. In that way an organization can properly assess and prioritize the treatment of its asset's vulnerabilities. The Vulnerability element shall have the following properties: id, name and description; an attack that describes how the vulnerability can be exploited; a score (e.g. in a 0 (low) to 10 (maximum) scale) that defines its severity and that may help to prioritize it; and a type.
The type of a vulnerability is based on CVE Details [15] classification of vulnerabilities which represents the "gains" of the attacker in case of success.

In addition, vulnerability has an attack vector (the route or path which an attack is carried out) that is classified in one of the following types, according the CVSS taxonomy [10]:

- *Network*: when the vulnerable component is bound to the network stack and the attacker's path is through OSI layer 3 (the network layer). An example of a network attack is when an attacker causes a denial of service (DoS) by sending a specially crafted TCP packet from across the public Internet.
- *Adjacent Network*: when the vulnerable component is bound to the network stack, however the attack is limited to the same shared physical network (e.g. Bluetooth, IEEE 802.11) or logical network (e.g. local IP subnet) and cannot be performed across an OSI layer 3 boundary (e.g. a router).
- *Local*: when the vulnerable component is not bound to the network stack and the attacker's path is via read/write/execute capabilities. In some cases, the attacker may be logged in locally in order to exploit the vulnerability; otherwise, it may rely on user interactions to execute a malicious file.
- *Physical*: when the attacker physically touches or manipulates the vulnerable component. An example is an attacker using a pen USB with malicious files directly into a USB port in a server room.
- *Organizational*: by manipulating the organization namely through its employees. An example could be an attacker kidnaping an employee with admin credentials and making a transfer of company money to his account.

4 CATALOGUE4CYBERSECURITY

In general a catalogue shall include multiple solution-oriented requirements because they tend to be reusable and applicable to different situations, this is the reason they are also known as "requirement patterns" [8,16,17]. The Catalogue4CyberSecurity has been developed as a set of RSL package files, i.e. as packages of reusable RSL specifications.

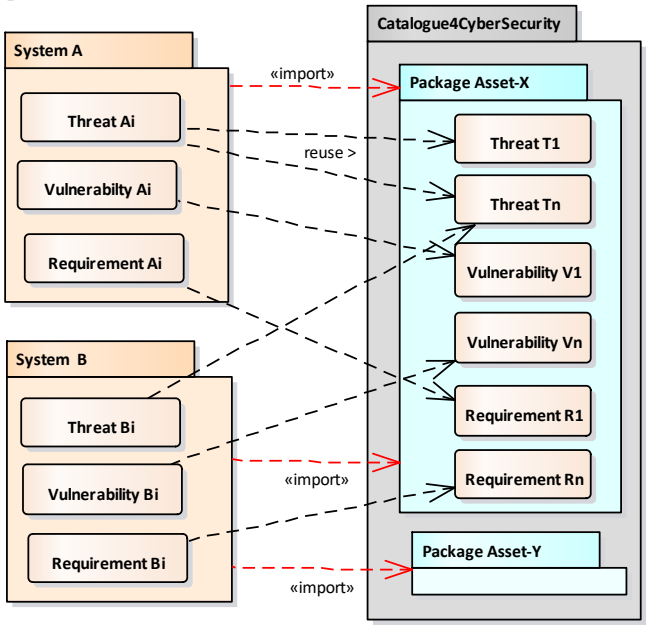

Fig. 3. Structure and use of the Catalogue4CyberSecurity.

As suggested in Figure 3 the Catalogue4CyberSecurity consists in a set of reusable RSL packages that can be imported and used by other system packages. Each package includes the specification of several elements (such as requirements, vulnerabilities, and threats) and respective relationships. Each package represents an *asset* that can be a concrete or a reusable system.

Table 4. Assets taxonomy.

Type	Sub-types
Application	Application_Web | Application_Mobile | Application_Desktop | Application_Sensor | Application_Actuator | Application_Other
SoftwareServer	SoftwareServer_VM | SoftwareServer_OS | SoftwareServer_DBMS | SoftwareServer_Web | SoftwareServer_Application| SoftwareServer_Email | SoftwareServer_Other
Device	Device_Server | Device_Desktop | Device_Laptop | Device_Smartphone | Device_Smartwatch | Device_Sensor | Device_Actuator |Device_Storage | Device_Printer | Device_Network |Device_Other

An asset is classified according the taxonomy shown in Table 4 suggesting common layers that exist in IT systems, namely: *Device* that represents the hardware infrastructure layer composed of computer servers, smartphones, desktops, sensors, etc.; *SoftwareServer* that

represents the software infrastructure layer composed of virtual machines, operating systems, application servers, etc.; and *Application* that represents the software application layer composed of multiple types of applications.

5 PRIVACY PACKAGE

This section presents an example of a RSL package that exists in the proposed catalogue: *Generic Web App's Privacy Concerns package*. This package is a collection of reusable requirements, vulnerabilities and threats that deal with privacy concerns commonly found in web applications [18]. This package includes the specification of concerns involving different levels of threats such as at user browser (e.g., Reflective XSS [19], or Man-in-the-Browser [20] attacks), communication between system components (e.g., TLS Protocol, HTTPS [24]), or even at the physical and software infrastructure levels like the database of the service provider (e.g. Information Storage [21] attacks). In that way this package considers the necessary components and layers of a web application.

This package is particularly focused on information managed in the scope of web applications, and so it is inspired on the concerns discussed by Deng et. al [3]. Deng's approach has the purpose of eliciting privacy requirements of software-intensive systems and select privacy enhancing technologies accordingly; it is known as the LINDDUN methodology in which each letter represents a type of privacy threat obtained by negating a corresponding privacy property. Table 5 summarizes these privacy properties with the respective threats.

The LINDDUN framework has properties similar to the RSL language, since these privacy properties are represented as System Goals in RSL while the privacy threats are represented as RSL's Risks. On top of the LINDDUN framework, RSL4Cybersecurity considers the identification and specification of vulnerabilities which makes the specification schema more comprehensive and flexible, also this package fulfils with at least one requirement and one threat for each of the LINDDUN.

Table 6 shows some examples of vulnerabilities that a system with privacy concerns might have, namely the "Man-in-the-Browser" and the "Cross-site Scripting network" vulnerabilities. On the other hand, Table 7 shows two examples of System Goals that a system with privacy concerns should adopt to protect itself, namely its needs for "Browser Up-to-date" and for "HTML XSS Prevention". Both examples are solution and classified as Security/Tampering requirements. Table 8 shows two examples of Threats that a system with privacy concerns may be subject to. The first threat is the "Confidential Information Stolen from the Browser" due to the vulnerability "Man-in-the-Browser", and is mitigated by the requirement "Browser Up-to-date"; the second threat is "User is Redirected to Malicious Website (Fishing)" due to the vulnerability "Cross-site Scripting" and is treated by the requirement "HTML XSS Prevention".

Table 5.LINDDUN concepts (adapted from [3])

Privacy properties	**Privacy threats**
Unlinkability	Linkability
Anonymity	Identifiability
Plausible deniability	Non-repudiation
Undetectability	Detectability
Confidentiality	Disclosure of information
Content awareness	Content Unawareness
Consent Compliance	Consent Noncompliance.

Table 6. Privacy Package: Vulnerabilities

Name (ID), ***Attack vector***	***Type*****,** **Attack**	**Description**
Man-in-the-Browser (vu_ man_in_the_ browser): *Network*	*Gain Information,* The attacker infects the user browser with a trojan horse that allows the attacker to modify and intercepted communication data.	Attacker copies confidential information from the browser [20].
Cross-site Scripting (vu_ cross_site_ Scripting): *Network*	*Cross-site scripting,* XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.	The attackers distract the user to a similar malicious website to steal personal information [19].

Table 7. Privacy Package: System Goals

Name (ID): ***Type***	**Description**
Browser Up-to-date (g_browser_up_to_date): *Security/Tampering*	Force the user to most up-to-date his browser version to prevent some vulnerabilities [20].
HTML XSS Prevention (g_html_xss_prevention) : *Security/Tampering*	Untrusted data must be placed in place where code executed is not allowed [19].

Table 8. Privacy Package: Threats

Name (ID): ***Type***		**Description**
Confidential Information Stolen from the Browser (th_browser_info_stolen): *Information disclosure* *LINDDUN:* Disclosure of info		Attacker steals information when the user inputs it in the browser [20].
Vulnerability	**Treatment**	
	Type	**Measure**
vu_ man_in_the_browser	*Mitigate*	g_browser_ up_to_date
Name (ID): ***Type***		**Description**
User is Redirected to Malicious Website (th_redirect_malicious_ website): *Information disclosure* *LINDDUN:* Disclosure of information		Attacker steals information when user inputs it in the browser after being redirected to a malicious website [19].
Vulnerability	**Treatment**	
	Type	**Measure**
vu_cross_site_ scripting	*Mitigate*	g_html_xss_ prevention

Table 9. Privacy Package: Number of elements per type.

Source	Threats	Vulnera-bilities	System Goals
Total Nº of Elements	9	11	16
Common Criteria [21]	-	-	16
FIRST [22]	-	3	-
OWASP [23]	9	8	-

Table 9 presents the size of the privacy package, i.e. the number of elements in its current version as publicly available [18]. The sources used to define this package (i.e., CC, FIRST and OWASP) are major references from the cybersecurity community.

6 Illustrative Example

This section shows how to reuse of the concerns introduced above by debating its application into a simple but illustrative example. This example is adapted from Deng et al. [3], and refers to a web application named "Social Network 2.0". This application is a social network that allows its users to share personal information like photos or comments. This example raises concerns regarding privacy and integrity of the involved information.

The specification of this app is defined in a specific package that uses/includes resusable packages (in particular the "privacy package" discussed in Section 5) which represent relevant assets. In this example the asset is the "Social Network 2.0" web app; due to space limitations we will simplify this system to just consider three components, one for each layer: the user-browser to server communication (Application layer), the DBMS (Software server) and the physical database (Device).

Tables 10, 11 and 12 show the specification of Vulnerabilities, System Goals and Threats that are defined to protect the user browser communication consider at the application level. In particular the threat "Confidential information stolen from eavesdropping the communications" is identified due to the vulnerability "Communication sniffing" and the propose approach to mitigate it, is to implement the requirement "Communication protection".

Tables 13, 14 and 15 show the concerns specification considered to protect this web application from unauthorized disclosure of information. This information is maintained by the MongoDB DBMS. Because of the use of the MongoDB it is identified the threat "MongoDB breached" due to the vulnerability "MongoDB Network Message Compressors"; To prevent this situation a solution is to adopt the requirement "Disable MongoDB network Message Compressors".

Table 10. User browser communication: Vulnerabilities

Name (ID): ***Attack vector***	***Type*****, Attack**	**Description**
Communication sniffing (vu_coms_sniffing): *Network*	*Gain Information,* The attacker uses a network sniffing tool to intercept the communication between two ends and to retrieve confidential information such as the credentials of the users	Attacker copies confidential information from intercepting communications. [24]

Table 11. User browser communication: System Goals

Name (ID): ***Type***	**Description**
Communication protection (g_coms_protect): *Security/Confidentiality*	This pattern shows how a browser creates an TLS connection with the server. [24]

Table 12. User browser communication: Threats

Name (ID): ***Type***	**Description**	
Confidential information stolen from eavesdropping the communications (th_eavesdrop_com): *Information disclosure* *LINDDUN:* Disclosure of information	Attacker eavesdrop confidential information from communications using a network sniffing program. [24]	
Vulnerability	**Treatment**	
	Type	**Measure**
vu_coms_sniffing	*Mitigate*	g_coms_protect

Table 13. Database DBMS: Vulnerabilities

Name (ID): ***Attack vector***	***Type*****, Attack**	**Description**
MongoDB Network Message Compressors (vu_MongoDB_ ntwrkMsgCompressors): *Network*	*Code Execution,* When wire protocol compression is enabled, a malicious attacker may exploit an existing vulnerability copy or modify server memory [23]	The wire protocol compression is enabled.

Table 14. Database DBMS: System Goals

Name (ID): ***Type***	**Description**
Disable MongoDB network Message Compressors (g_MongoDB_msg_compression): *Security/Tampering*	Protect MongoDb database by disabling the wire protocol compression. [23]

Table 15. Database DBMS: Threats

Name (ID): ***Type***	**Description**	
MongoDB breached (th_mongodb_dos): *Confidentiality* *LINDDUN: Confidentiality*	Attacker using wire protocol compression accesses database data. [23]	
Vulnerability	**Treatment**	
	Type	**Measure**
vu_MongoDB_ ntwrkMsgCompressors	*Mitigate*	g_MongoDB_msg_compression

Table 16. Physical Database: Vulnerabilities

Name (ID): ***Attack vector***	***Type*****,** **Attack**	**Description**
Database USB ports unprotected (vu_USB_port_DB): *Physical*	*Gain Information,* The attacker inserts an USB pen into the server and copies company confidential information from its users.	The USB ports in the server are enabled and unprotected. [23]

Table 17. Physical Database: System Goals

Name (ID): ***Type***	**Description**
Server USB port protection (g_usb_protect): *Security/ Tampering*	Protect server USB ports by disabling executables and copy of files without admin credentials. [23]

Table 18. Physical Database: Threats

Name (ID): ***Type***	**Description**	
Confidential information stolen from DB through unprotected USB port (th_db_port): *Information disclosure* *LINDDUN:* Disclosure of information	Attacker copies confidential information from server using unprotected USB port. [23]	
Vulnerability	**Treatment**	
	Type	**Measure**
vu_USB_port_DB	*Mitigate*	g_usb_protect

Finally, Tables 16, 17 and 18 show the concerns specification considered to protect the web application from unauthorized access. In particular, it is identified the threat "Confidential information stolen from DB through unprotected USB port" due to the vulnerability "Database USB ports unprotected", and one possible solution to prevent this situation is to implement the requirement "Server USB port protection".

From the analysis of this web app, we show how to better define these packages of reusable specifications, and how to better structure the proposed Catalogue4Cybersecurity. Its users should start by choosing the package(s) that shall fit their specific needs, and from them they may have a series of requirement, threats and vulnerabilities already well defined and prompted to be reused in their own specifications.

7 Related Work

In the scope of requirements engineering (RE) some researchers have proposed techniques to help specify security requirements in a slightly systematic way [2,3,4,5,16]. While there is not a research proposal similar to ours, there were some that help building our proposal. These techniques tackled the elicitation and specification of security requirements since those were the tasks that would be most fitted with our objectives.

Ferreira and Silva [7] proposed initially an interesting and ambitions approach to deal with requirements. Recently in this scope Silva presented a large RE-specific language, called RSLingo RSL [8,12] and supported by a companion Excel template publicly available [11]. RSLingo RSL is a comprehensive language for requirements specification but it is limited in what concerns the security requirements specification; hence there is a real need to extend this language with these concerns.

Sindre et al. [4] extended the traditional use case concept to consider *misuse cases*, which represent behaviour not wanted in the system to be developed. Likewise, McDermott et al. [9] introduced the concept of *abuse cases* as a set of interactions between one or more actors and a system, where the results of these interactions are harmful to that system. Both modelling techniques have a significant benefit since developers who work on the security features of such software systems do not understand mathematical security models. These concepts were useful because they are close to the classical *use cases* as they allow specifying unwanted behaviour, i.e. in what we called as *negative requirements*.

Myagmar et al. [5] investigated how *threat modelling* can be used as a foundation for the specification of security requirements. To have a reliable threat model, it cannot be created by simply brainstorming an adversary's possible intentions because, in that way, it is likely to leave large portions of the attack-space un-investigated. An attacker only has to find one security flaw to compromise the whole system. Thus, it is important to be systematic during the threat modelling process to ensure that as many possible threats and vulnerabilities are discovered by the developers, not the attackers. The *threat model process* proposed by Myagmar et al. has three steps: characterizing the system, identifying assets and access points, and identifying threats. In the first step, the system must be characterized preferably by a data flow diagram which dissects the application into its functional components and indicates the flow of data in and out of the various system components. The second step, using the data flow diagram, the assets and access points are identified. And in the third step, the threats are identified. This systematic method was a good starting model to adapt in our work, namely by considering the concepts of assets and threats.

Mouratidis and Giorgini [2] proposed Secure Tropos, an extension to Tropos methodology, which allows considering security issues throughout the development process of multiagent applications. Tropos adopts the i* modelling framework with concepts of actors, goals, tasks, resources, and social dependencies for defining the obligations between actors. To enable developers adequately capture security requirements they introduced the concept of constraint and discussed how to extend the Tropos concepts with security in mind.

Deng et al. [3] proposed a comprehensive framework and a systematic methodology to model *privacy-specific*

threats known as the LINDDUN methodology. That framework includes seven privacy threat types: unlikability, anonymity and pseudo-anonymity, plausible deniability, undetectability and unobservability, confidentiality, user content awareness, and policy and consent compliance. Moreover, based on this framework, they provided a set of privacy-specific threat tree patterns that can be used to support a threat analysis.

FIRST [22] is an international confederation of trusted computer incident response teams who cooperatively handle computer security incidents and promote incident prevention programs. FIRST offers a set of practices and tools to help characterize vulnerabilities, namely the "Common Vulnerability Scoring System" (CVSS) [10] that provides a way to capture the characteristics of vulnerabilities, and produce a numerical score reflecting their severity.

The Common Criteria for Information Technology Security Evaluation (Common Criteria or just CC) [21] is an international standard for computer security certification which assures that the process of specification, implementation and evaluation of a computer security product has been conducted in a rigorous, standard and repeatable manner at a level that correspond with the target environment for use. By using standards like CC in the design and evaluation of Catalogue4CyberSecurity, it is assured that the *system goals* used are vastly tested, secure and reliable, has these processes are certified by the numerous country members.

Microsoft also defined a process to help developers build more secure applications named as Security Development Lifecycle (SDL) [26]. This process includes the following phases: Requirements (security requirements are established); Design (the attack surface and threat modelling are analysed); Implementation (the system is implemented); Verification (the system is tested); and Release (the software is released). The design phase was the most significant contribution as it includes the threat modelling and analysis. This phase incorporates four steps: Diagram (create a data flow diagram of the system); Identify threats (using the data flow diagram and the STRIDE [14] model to identify threats); Mitigate (address each threat identified); and Validate (validate the whole threat model).

Beckers et al. [16] proposed a catalogue of security and privacy requirement patterns that would support software engineers in their task of eliciting security requirements. According to them a *security pattern* is a guideline to produce a secure communication channel between two entities and it is a general reusable solution to a commonly occurring problem in a software design domain. Beckers et al. showed how security requirements can be classified according to cloud security and privacy goals. For that purpose, they defined the "ClouDAT framework" that allow eliciting security requirements of cloud-based systems. That framework includes a meta-model of a cloud system and an associated context-pattern and templates. This research was useful in the definition and validation of our own catalogue.

At a more concrete level Stell et al. [17] discussed practices and strategies to deal with and implement security in Java based web applications using an extensive number of security patterns, which we consider at a more concrete and solution level.

Firesmith et al. [6] suggested using textual security requirements templates to make security requirements highly reusable. They provided an asset-centered and reuse-based procedure for requirements and security teams to analyse security requirements involving 13 steps, starting by identifying the valuable assets, identifying threats, and estimating vulnerabilities. These steps end by specifying requirement through the instantiation of templates based on the parameters from the previous steps.

Caramujo and Silva [27] propose the RSL-IL4Privacy, a domain-specific language defined originally as a privacy-aware profile that identified the main concepts of current privacy policies. RSL-IL4Privacy allows to specify privacy policies by providing constructs such as data type, data recipient and enforcement mechanism, which are necessary to specify and document privacy-related requirements. However, it is targeted to just privacy policies and not support the general specification of security concerns.

FIRST [22], Deng et al. [3], Microsoft SDL [22], Myagmar et al. [5] were important references that inspired our research since they help to understand and gather the necessary concepts for the specification of vulnerabilities and threats. The contributes of Firesmith et al. [6], Beckers et al. [16], and Stell et al. [17] guided us to define the proposed catalogue structure as they included example of templates and properties needed to properly characterize security requirements.

8 Conclusion

Cyber-security is a huge concern in our society that may be analyzed from different perspectives, namely from personal, organization or even government perspectives. However, regrettably many organizations still tend to realize the importance of security only after actually suffering attacks and get important damages. Even when security is understood as a key concern, too often it is only considered into the systems only at the end of their construction phase which leads to many vulnerabilities and consequent negative impacts.

It is consensual that security is a cross-cutting concern that shall be considered at multiple levels of the systems:

from hardware and software infrastructures to business and organizational policies and practices, but also at application system level. Thus, it is also consensual that these concerns shall be identified and specified since the very beginning of these systems implementation and deployment, namely in the preliminary requirements engineering activities. However, in spite of some approaches that have been proposed recently [4], [5], [6], [8], to the best of our knowledge this is the first work that, by extending an existent RSL language, provides a rigorous and systematic approach to specify not only different types of requirements (e.g., security patterns, quality requirements, misuse cases, or user stories) but also other security-specific constructs such as threats, vulnerabilities and respective relationships.

In addition, on top of this extended language we also show that it is possible to develop and use a catalogue of reusable security-specific specifications. This catalogue shall be progressively developed with an increasing number of RSL packages. Each package gathers all the specifications of requirements, threats, vulnerabilities, etc. that would be relevant for a particular type of system or a class of systems. For example, it should be possible to define a package for a particular version of an operating system, data base server, or office suite; or a package for a more general class of systems such as Web apps from a privacy perspective (like it was discussed in the paper), or more specific like for Java-based web applications.

For future work we want to extend the RSL language with more references and concerns and also with the possibility to include rigorous specification of security acceptance and penetration tests [28], e.g., based on the linguistic pattern "given-when-then" popular in BDD approaches [29]. Moreover, we intend to disseminate the Catalogue4CyberSecurity with its multiple packages. We also intend to integrate it with other security frameworks and processes such as the Microsoft SDL [26] to increase the efficiency of these processes with the aid of reusable requirements. Another feature that might be included is the support of dynamic classification schemas.

ACKNOWLEDGEMENTS

This work was partially supported by national funds under FCT project UID/CEC/50021/2013.

REFERENCES

[1] OWASP, *Web Application Firewall*, www.owasp.org/index.php/Web_Application_Firewall. 2018.

[2] H. Mouratidis, and P. Giorgini, "Secure tropos: a security-oriented extension of the tropos methodology", *International Journal of Software Engineering and Knowledge Engineering*, 2007.

[3] M. Deng, K. Wuyts, R. Scandariato, B. Preneel, and W. Joosen, "A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements", *Requirements Engineering*, Springer, 2011.

[4] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases". *Requirements Engineering,* 10.1, 2005.

[5] S. Myagmar, A. J. Lee, and W. Yurcik: "Threat Modeling as a Basis for Security Requirements", *Symposium on requirements engineering for information security*, 2005.

[6] D. Firesmith, "Specifying reusable security requirements", *Journal of Object Technology*, 2004.

[7] D. Ferreira, A. R. Silva, "RSLingo: An information extraction approach toward formal requirements specifications", *Proceedings of MoDRE'2012*, IEEE CS, 2012.

[8] A. R. Silva, "Linguistic Patterns and Styles for Requirements Specification: The RSL/Business-Level Language", *Proceedings of EuroPLOP'2017*, ACM, 2017.

[9] McDermott, J. and Fox, C.: "Using abuse case models for security requirements analysis". In *Computer Security Applications Conference*, IEEE, 1999.

[10] FIRST - Forum of Incident Response and Security Teams, *Common Vulnerability Scoring System*, www.first.org/cvss. 2018.

[11] ITLingo - *Specification Languages for the IT domain*, https://www.researchgate.net/project/ITLingo-Specification-Languages-for-the-IT-domain.

[12] A. R. Silva, "A Rigorous Requirement Specification Language for Information Systems: Focus on RSL's Use Cases, Data Entities and State Machines", *INESC-ID Technical Report*, 2017.

[13] Eclipse Xtext, www.eclipse.org/Xtext, 2018.

[14] The STRIDE Threat Model, msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx. 2018.

[15] CVE Details, *The ultimate security vulnerability data source*, https://www.cvedetails.com/

[16] K. Beckers, I. Côté, and L. Goeke: "A catalog of security requirements patterns for the domain of cloud computing systems", *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. ACM, 2014.

[17] C. Stell, "*Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*", Prentice Hall PTR, 2005.

[18] L. Gonçalves, A. R. Silva, "*Generic Web App's Privacy Concerns (RSL Package)*", available in [11], 2018.

[19] OWASP, *Cross-site Scripting (XSS),* https://www.owasp.org/index.php/Cross-site_Scripting_(XSS), 2018

[20] OWASP, *Man-in-the-browser attack,* https://www.owasp.org/index.php/Man-in-the-browser_attack, 2018.

[21] Common Criteria, www.commoncriteriaportal.org. 2018.

[22] FIRST, *Forum of Incident Response and Security Teams*, www.first.org. 2018.

[23] OWASP, https://www.owasp.org, 2018

[24] OWASP, *Transport Layer Protection Cheat Sheet,* https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

[25] OWASP, *Sniffing application traffic attack*, https://www.owasp.org/index.php/Sniffing_application_traffic_attack

[26] Microsoft, *Security Development Cycle*, https://www.microsoft.com/en-us/SDL. 2018.

[27] J. Caramujo, A. R. Silva, "Analyzing Privacy Policies based on a Privacy-Aware Profile: the Facebook and LinkedIn case studies", Proceedings of IEEE Conference on Business Informatics'2015, IEEE, 2015

[28] A. R. Silva, A. R. Paiva, V. R. Silva, "Towards a Test Specification Language for Information Systems: Focus on Data Entity and State Machine Tests", Proceedings of MODELSWARD'2018, SCITEPRESS, 2018.

[29] Wynne, M., Hellesoy, A., Tooke, S.: "The cucumber book: behaviour-driven development for testers and developers". Pragmatic Bookshelf, 2017.