Towards a Catalogue of Reusable Security Requirements, Risks and Vulnerabilities

Luís Pissarra Gonçalves *luiscpgoncalves@tecnico.ulisboa.pt*
INESC-ID, Instítuto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal

Alberto Rodrigues da Silva *alberto.silva@tecnico.ulisboa.pt*
INESC-ID, Instítuto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal

Abstract

Organizations are giving more importance to secure their systems due to the increasing number of cyber-attacks and inherent complexity. The aim of our work is help organizations plan and consider these security concerns from the very beginning, since the requirements and design phases, and not just later in the implementation or deployment phases. Consider security-by-design and security-by-default principles are good approaches to avoid rework costs or to mitigate security flaws. However, there is not yet a suitable approach to specify security requirements in a rigorous and systematic way. In this paper we propose an approach that allows the definition and specification of security-specific concerns like security requirements but also vulnerabilities, risks or threats. We discuss this approach based on two key parts: First, we introduce the RSLingo RSL language, that is a rigorous requirements specification language, and discuss how it is extended to support such security-specific concepts. Second, we claim the relevance for a catalogue of reusable security-specific specifications and then we show concrete examples of defining and using such specifications. The proposed catalogue can be easily used and extended by the community and involves currently 52 goals, 12 vulnerabilities and 31 risks; these concerns are defined into 9 packages each one representing a distinct asset.

Keywords: Requirements specification, Cyber-security, Catalogue of security requirements

1. Introduction

System attacks and breaches have increased throughout the recent years. Organizations are struggling to keep their systems protected from attacks that are constantly getting bigger and more complex. Security shall be considered and incorporated into such systems right from the beginning of the design phase, so that vulnerabilities and negative impacts are minimized [6], [15]. Some approaches have helped to improve the threats elicitation and modelling of such situations [12], [17], including models and frameworks to support security specifications [10, 11], [18]. But still, there is not a consensus on which solution to adopt or which method provides the best results. Moreover, these approaches do not consider the entire problem: they only address a partial perspective of the whole picture. For instance, there has been proposals for extensions or modifications to traditional methods, namely use cases [17], [12]. However, these proposals do not relate practices and tools from important references from the cyber-security community such as FIRST [9] or SDL [14].

In this research we extend the RSLingo RSL language [13], [18] 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 or risks), we explain how to extend it for this purpose. We follow a pragmatic approach to implement such extension, namely by directly update the RSL Excel template followed by the ITLingo Studio publicly available [13]. For the sake of simplicity, in the context of the paper, we named “RSL4Cybersecurity” as this RSL language extended version.

Furthermore, we also discuss the definition and structure of a catalogue of security-specific concerns. The objective of this catalogue is to aggregate multiple packages with reusable specifications. In this way, known problems and well-tested solutions to generic systems are easily accessible and reused by concrete system specifications. Again for the sake of simplicity, in the context of the paper, we named "Catalogue4CyberSecurity" this set of well-defined packages of reusable specifications.

This paper is structured in 7 sections. Section 2 introduces and discusses the related work that influenced our research. Section 3 briefly introduces the RSL language that was adopted to support the specification of security-specific concerns. Section 4 discusses the RSL extensions proposed as a set of additional requirement properties as well as new security concepts. Section 5 discusses the proposal of a catalogue of RSL packages with reusable security requirements and related elements, including risks and vulnerabilities. Section 6 discusses the current version of the Catalogue4CyberSecurity that has being developed considering popular references from the community. Lastly, Section 7 presents the conclusion and identifies open issues for future work.

2. Related Work

In the domains of cyber-security and requirements engineering (RE) researchers have proposed techniques to help specify security requirements in a slightly systematic way [1] ,[7],[11],[16,17]. Our research has been mostly influenced by these proposals, particularly focused on elicitation and specification of security requirements, since these were the tasks that would be most fitted with our objectives.

Ferreira and Silva [11] 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* [18]. RSL is supported by the ITLingo-Studio, an Eclipse-based tool for IT technical documentation, and it is also supported by a companion Excel template publicly available [13]. RSL is a comprehensive language for requirements specification but it is limited in what regards the security requirements specification; hence there is a real need to extend RSL with these concerns (Section 3 details the RSLingo RSL language).

Sindre et al. [17] 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. [12] 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 behavior, i.e. in what we called as *negative requirements*.

Myagmar et al. [16] studied how *threat modelling* can be used as a foundation for the specification of security requirements. They highlight the fact that we need a reliable threat model, since a simple flaw in the system can compromise the system has a whole. 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.

Deng et al. [6] 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: unlinkability, anonymity and pseudonymity, 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 [9] is a 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) [8] that provides a way to capture the characteristics of vulnerabilities, and produce a numerical score reflecting their severity.

CVE Details [4] is a database of vulnerabilities from various products, vendor and version which aim is to provide an easy to find list of known vulnerabilities. These vulnerabilities are categorized using the FIRST CVSS scoring system.

Mouratidis and Giorgini [15] 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.

Microsoft also defined a process to help developers build more secure applications named as Security Development Lifecycle (SDL) [14]. This process includes the following phases: Requirements (security requirements are established); Design (the attack surface and threat modeling are analyzed); Implementation (the system is implemented); Verification (the system is tested); and Release (the software is released). The design phase is the most significant contribution as it includes the threat modeling 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 [23] model to identify threats); mitigate (address each threat identified); and validate (validate the whole threat model).

Firesmith et al. [10] 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 analyze 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 [19,20] 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 specifying 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.

Beckers et al. [1] 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 technical level Stell et al. [21] 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.

FIRST [8], CVE Details [4], Deng et al. [16], Myagmar et al. [16] were important references that inspired our research since they help us to understand and gather the necessary concepts for the specification of vulnerabilities and risks. The contributes of Firesmith et al. [10], Beckers et al. [1], and Stell et al. [21] guided us to define the security catalogue structure as they propose templates and properties needed to properly characterize security requirements.

3. RSLingo RSL

RSLingo is a research initiative in the RE area that recognizes that natural languages, although being the most common and preferred form of representation used within requirements documents, tend to produce ambiguous and inconsistent documents that are hard to automatically validate or transform [6]. Recently, it has been proposed a broader and consistent language, called "RSLingo RSL" (or just "RSL" for brevity), based on the design of former languages [18]. RSL is a controlled natural language designed to help the production of requirements specifications in a systematic, rigorous and consistent way [18]. RSL includes a rich set of constructs (e.g. stakeholders, actors, data entities, requirements) logically arranged into views according to RE-specific concerns that exist at different system levels. These constructs are also defined as linguistic patterns and represented textually by mandatory and optional fragments [18].

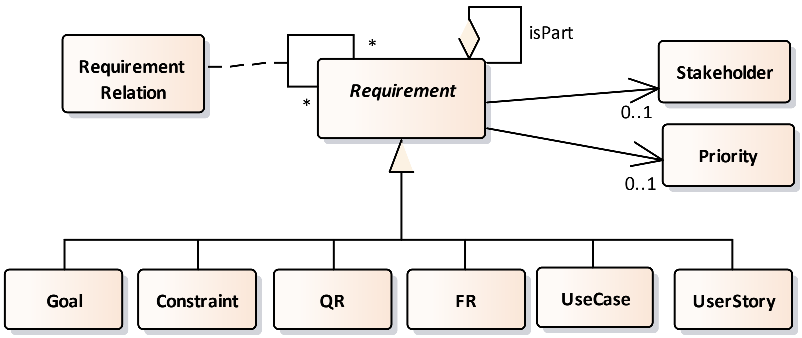


Fig. 1. RSL partial metamodel: the hierarchy of requirements.

Fig. 1 shows a partial view of the RSL metamodel, that defines a hierarchy of requirement types, such as goals, functional requirement, constraint, use case, user story or quality requirement. 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; these relationships can be further classified as Requires, Supports, Obstructs, Conflicts or Identical. RSL has been implemented with the Xtext framework [7] in the scope of an Eclipse-based tool called ITLingo-Sudio [13]. Consequently, RSL specifications are rigorous and can be automatically validated and transformed into other representations and formats. Furthermore, it is also provided an RSL Excel template, publicly available at ResearchGate [13]. In spite of the large set of constructs, RSL lacks support for security requirements specification; for instance, the previous RSL version allowed to specify security requirements as quality requirements (including a quality sub-type with values like confidentiality or authorization), but there were no constructs to specify risks, vulnerabilities and the respective relationships.

4. RSL Extension for Security

There are some concepts and terms that emerged from the literature analysis when we try to specify and analyze system's security concerns. As introduced in Section 2, some of these concepts are vulnerabilities, attacks, threats, risks, treatments measures, requirements, goals, misuse cases and abuse cases. This section discusses how to define and relate these common concepts. To support the discussion, we introduce a set of sentences that exemplifies hese concerns: sentences S1 to S6 in Table 1.

Each RSL requirement has its own classification schema; for example, goals can be further classified as security performance, usability etc. However, from the analysis of the related work (see Section 2) and from our preliminary attempts to specify reusable requirements, we observed the urge for some extension schema to better classify such requirements. This RSL extension starts by redefining the Requirement concept. We add three new properties to define a requirement: *isPositive*, *isAbstract* and *isProblem*. By default, and if not explicitly stated, a requirement shall be classified as *positive, abstract and problem-oriented*. Table 2 classifies the sentences introduced in Table 1, according to this schema.

Table 1. Example of sentences informally describing security concerns.

S1	System shall guarantee that critical data saved at DBs or file system are kept private by some encryption technique.
S2	System shall guarantee that critical data saved at DBs or file system are kept encrypted with the Data Encryption Standard (DES).
S3	A vulnerability in the MySQL Server database could allow a remote, authenticated user to inject SQL code that MySQL replication functionality would run with high privileges. A successful attack could allow any data in a remote MySQL database to be read or modified.
S4	The Secure Logger pattern provides instrumentation of the logging aspects in the front, and the Audit Interceptor pattern enables the administrator and manages the logging and audit in the back-end.
S5	An attacker can use vulnerabilities of MySQL to access and modify a database.
S6	John (a user without admin access) uses his credentials in the database terminal and modifies users permissions (operation only available with admin credentials).

The *isPositive* property allows to state if the requirement is positive or negative. A *positive requirement* 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 confidential”] refers a confidential-related goal in an abstract way; while S2 [“data…are kept encrypted with the Data Encryption Standard (DES)”] extends the previous requirement with more specific information. The *isProblem* property allows identifying if the requirement is stated as a problem or as a solution. In general, a requirement is stated as a problem and consequently is focused on the *what* of the system; that is the common situation, for example of *use cases* based specifications. However, in other situations, we want that requirements explicitly express the *solution* (i.e., the *how*) for an identified problem. We also may adopt the expression “*requirement pattern*” for such solution-oriented requirements. An example of a solution requirement is S4 that recommends the adoption of the Secure Logger pattern as a secure management solution regarding the logging and audit features of Java based web applications.

In the context of security requirements specification there are concepts that shall not be “requirements” because they do not represent needs or wishes from the stakeholders. For instance, the sentences S3 and S5 are examples of such information that namely refer vulnerabilities (S3), threats or risks (S5). Consequently, some questions appear when we intend to refer these new concepts, for instance: what properties shall be added for defining them? what classification schema shall be adopted?, or how shall we relate requirements with risks and vulnerabilities?

Figure 2 shows the partial meta-model of the RSL security-specific extension. The new concepts are Risks and Vulnerabilities. A *Risk* is an event that can happen, usually with a negative consequence; a risk can be associated to several well-known vulnerabilities of some system. In addition, it is possible to define the risk assessment level, and treatment measures. Each treatment measure is classified by a treatment strategy (e.g. avoid, mitigate, transfer or accept) and can be assigned to a security-specific requirement, preferentially a solution requirement that shall be previously defined. (In the scope of this paper “Risk” and “Threat” are synonyms. In spite of the term “threat” is commonly used in the security community, we adopted the term “risk” because it tends to be used in more general situations.)

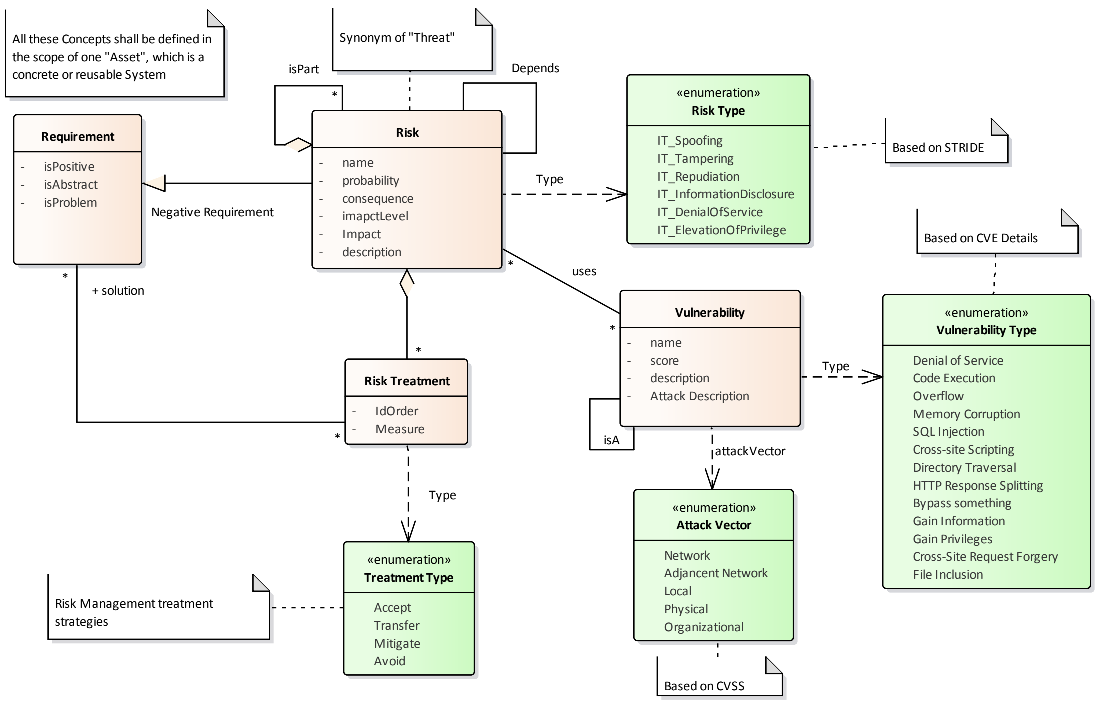


Fig. 2. RSL4CyberSecurity metamodel (partial view).

4.1. Risk Classification

A *risk* is the possibility of harming the system under analysis. We adopt the STRIDE threat taxonomy [23] to identify security risk types. STRIDE is an acronym for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege. These risks are the negation of the security properties, namely 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.

4.2. Risk Treatment

The severity of a risk shall be defined to prioritize and treat it accordingly; for that purpose some risk impact model shall be considered. Common risk impact models consider as the input the *likelihood* and the *consequence* of the risk and produce the *risk impact level* with the mean value of the two previous inputs. After defining the severity, it is possible to determine how to treat each risk according to the following risk treatment strategies:

- *Accept*, when the risk impact is so low that it is worth just accepting it;

- *Transfer*, when transfer the risk to some other entity via insurance, contracts etc;
- *Mitigate*, when reduce the risk impact level with countermeasures, such as requiring a long password, an encrypted communication;
- *Avoid*, when remove the system component or feature associated with the risk because the risk impact is too high and the organization cannot cope with it.

After identifying the best treatment can best address the risk, the analyst can also specify the concrete measure or solution for that problem; for instance, a system goal can be assigned as a convenient solution to treat a given risk.

4.3. Vulnerabilities

A *vulnerability* is a flaw in the system under consideration that opens it to an attack or exploit which will lead to damage or loss of property. The Vulnerability added to RSL is based on the *Common Vulnerability Scoring System* (CVSS) [8]. CSVV gathers 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 RSL Vulnerability construct has the following properties: id, name and description; attack that describes how the vulnerability can be exploited; and score, e.g. in a 0 (low) to 10 (maximum) scale, that identifies its severity and may help prioritize it. Furthermore, a vulnerability it is classified in one of the following types, accordingly with the CVSS taxonomy [8]:

- *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 occurs via read/write/execute capabilities. In some cases, the attacker is logged in locally to exploit the vulnerability; in other situations, he may rely on user interactions to execute a malicious file.
- *Physical*: when the attacker physically manipulates the vulnerable component. An example is an attacker using a pen with malicious files directly into a USB port.
- *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.

5. RSL Catalogue of Security Concerns

A catalogue (or library) of RSL specifications is organized as a hierarchy of folders with a set of files (or packages in the RSL terminology) of reusable systems. For simplicity reasons, an RSL package defines the specifications of just one system.

To better support the discussion we introduce a running example concerning the specification of an ATM system. ATMs (Automated Teller Machines) involve considerable amounts of cash and process sensitive customer data to perform cash transactions and banking operations. There are more than 3 million ATMs around the world, and they have been traditionally attacked physically by bandits trying to steal the cash inside [2]. Nowadays, criminals increasingly use logical attacks to manipulate ATM's software in order to withdraw cash or to capture customer data. The package that we introduce below will deal with both of these security aspects.

Figure 3 suggests the specification of security-specific elements for the concrete system Cash_Connect_ATM that reuse specifications defined by the reusable system Generic_ATM, which itself also reuse specifications of other general systems called DBMS, APP_layer and Physical_DB. In addition, Specifications 1, 2 and 3 show snippets of RSL specifications.

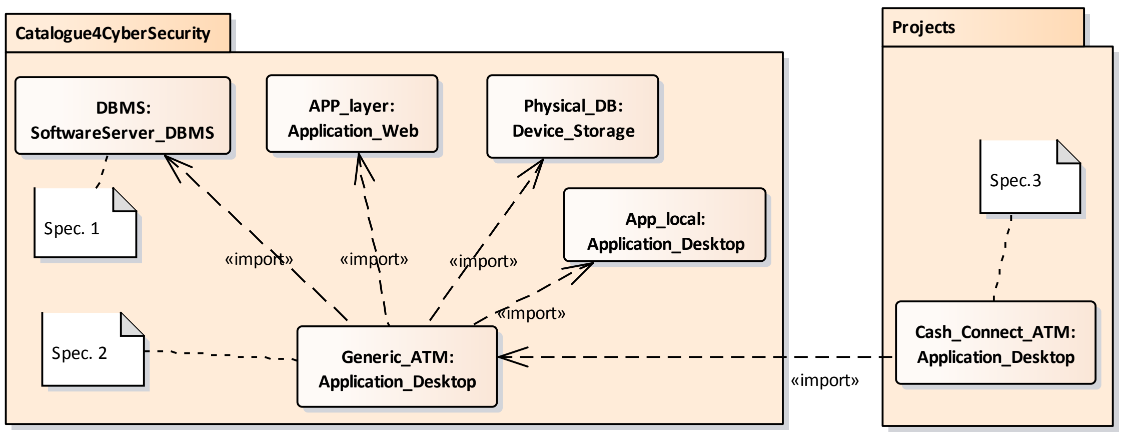

Catalogue4CyberSecurity
DBMS: SoftwareServer_DBMS
APP_layer: Application_Web
Physical_DB: Device_Storage
App_local: Application_Desktop
Spec. 1
Spec. 2
«import»
«import»
«import»
«import»
Generic_ATM: Application_Desktop
«import»
Projects
Spec.3
Cash_Connect_ATM: Application_Desktop

Fig. 3. Catalogue4CyberSecurity structure and Projects structure.

Specification 1 shows the specification of the generic DBMS system, with common vulnerabilities, risks and goals that exist in this class of DBMS systems. Specification 2 shows the specification of a generic ATM system. This ATM system imports three packages and reuses the elements from those systems to create a generic specification of the ATM. Finally, Specification 3 shows the specification of the concrete system Cash_Connect_ATM, which includes all the elements from the Generic_ATM package and specifies additional elements specific to the CashConnect ATM system.

5.1. Systems

As suggested in Figure 4, a System aggregates the definition of several SystemElements, which can be Requirements, Risks, Vulnerabilities, but also Actors, Includes, etc. A system represents an IT asset or a business as a socio-technical system, and so, a system is classified by a *type* suggesting common layers that exist at both business and IT levels, namely: Business, Application, SoftwareServer, and Device. Additionally and optionally, a system can be further classified by a *subtype* with values like: Business_PublicSector, Business_IT, Business_FinanceAndBanks, … Application_Web, Application_Desktop. *Business* system represents the business or organizational layer composed of organizational structures, business processes, business resources, etc. *Application* represents the software application layer composed of multiple types of applications. *SoftwareServer* represents the software infrastructure layer composed of virtual machines, operating systems, database servers, application servers. *Device* represents the hardware infrastructure composed of computer servers, smartphones, desktops, sensors, etc. In the ATM example, Cash_Connect_ATM and Generic_ATM are Application_Desktop systems, and DBMS is a SoftwareServer_DBMS.

Moreover, a system is also classified by another property: *isConcrete/isReusable*. By default, if not explicitly expressed, a system is defined as a concrete system. The *isConcrete* property states the system as some final system, meaning that there elements are not reusable. Oppositely, the *isReusable* means that such system represents a generic class of systems, and so, its elements can be reusable by other (either concrete or reusable) systems. In the ATM example, Cash_Connect_ATM is concrete while the other systems are reusable systems.

5.2. Imports, References and Includes

When defining a system, if we want to reuse elements defined in other system, first we have to explicitly define an import dependency for that system, and so we gain access to the elements defined in it. Therefore, we may reuse these elements adopting one of the following approaches: *Access by Reference*: The elements defined in our source system can reference any element defined in the imported systems. *Access by copy*: Alternatively to access by reference, we may decide to include in the specification of our source system all or some specific elements defined at the imported systems; for that purpose we shall use, respectively, the RSL *IncludeAll* or *Include* elements. The use of include elements implies the generation of a new system specification version where all these include elements shall be replaced by their original elements, yet with the possibility to replace some textual parts.

In the ATM example, the specification of the GenericATM system has several include elements, namely the element DBMS_g_credentials_rules that includes the goal g_credentials_rules defined in the DBMS system. The specification of the CashConnect_ATM system adopted a simple strategy by just include all the elements defined in the GenericATM system with the IncludeAll element.

5.3. Catalogue4CyberSecurity

The Catalogue4CyberSecurity has been developed as a set of RSL files. Figure 5 illustrates the main folder hierarchy as managed in the scope of the ITLingo-Studio tool. The top-level folder “Catalogue” has sub-folders containing reusable specifications for different domains, namely for cybersecurity: the “Catalogue4CyberSecurity” folder. This folder includes generic security packages organized by system classes, such as SmartToys, DBMS, Physical_DB. The other top-level folder is “Projects” where concrete systems are specified, e.g. CashConnect ATM, SmartToys.

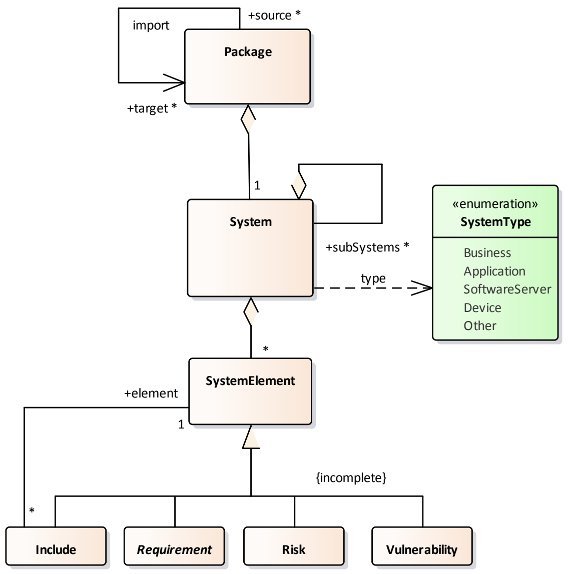
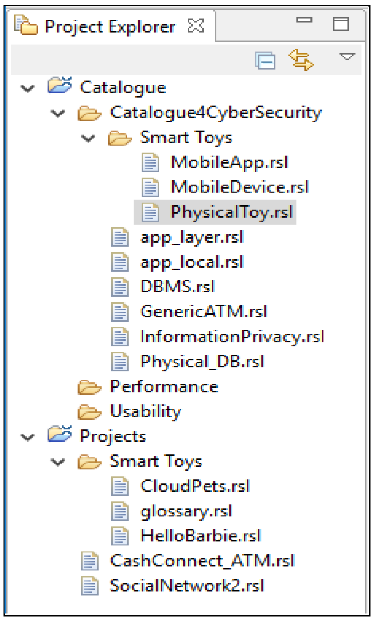


Fig. 4. Package, System and SystemElements.



```
Catalogue
  Catalogue4CyberSecurity
    Smart Toys
      MobileApp.rsl
      MobileDevice.rsl
      PhysicalToy.rsl
    app_layer.rsl
    app_local.rsl
    DBMS.rsl
    GenericATM.rsl
    InformationPrivacy.rsl
    Physical_DB.rsl
  Performance
  Usability
Projects
  Smart Toys
    CloudPets.rsl
    glossary.rsl
    HelloBarbie.rsl
  CashConnect_ATM.rsl
  SocialNetwork2.rsl
```

Fig. 5. Catalogue structure.

```
Package Catalogue4CyberSecurity.DBMS

System DBMS: SoftwareServer: SoftwareServer_DBMS [ isReusable ]

Goal g_credentials_rules "Credentials rules": Security: Security_Integrity [
   isPositive isConcrete isSolution
   description "Do not allow short or no-length passwords and do not apply character set, or encoding
restrictions on the entry or storage of credentials. Continue applying encoding, escaping, masking,
outright omission, and other best practices to eliminate injection risks. Maximum lenght 160.
This way we can difficult brute force attacks"]

Vulnerability vu_weak_credentials "weak credentials" :IT: IT_GainPrivileges[
   attack "With weak credentials an attacker guesses the password with brute force attacks or using
dictionary attacks"
   attackVector IT_Network
   score 5.0 ]
Risk th_credentials_guessing "Credentials guessing": IT: IT_InformationDisclosure[
   isNegativeRequirement g_credentials_guessing
   vulnerabilities vu_weak_credentials
   treatment T1:Mitigate[solution g_credentials_rules]
   treatment T2:Mitigate[solution g_pass_attempts
      description "block an account temporary after 4 attempts to login with wrong credentials"]
   description "The credentials of a user are discovered"]
```

Spec. 1. DBMS reusable system: RSL specification.

```
Package Catalogue4CyberSecurity.Generic_ATM

Import Catalogue4CyberSecurity.*

System GenericATM: Application: Application_Desktop [ isReusable version "1.0" description " ATMs mantain
considerable amounts of cash and process sensitive customer data to perform cash transactions and …"]

Include DBMS_g_credentials_rules: Goal from system DBMS element g_credentials_rules

Goal g_pass_policy "Password policy": Security: Security_Authentication[
   isPositive isAbstract isSolution
   partOf DBMS_g_credentials_rules
   description "Hamper pin guessing by blocking the user account after 3 consecutive wrong pin entries"]

Include g_usb_protect: Goal from system physical_db element g_usb_protect

Goal g_hd_encrypt "Full Hard Disk Encryption" :Security:Security_Confidentiality[
   isPositive isAbstract isSolution
   description "Physically protecting the hard disk is an additional safeguard because data access becomes
more difficult if HD stolen."]

Include vu_USB_port_DB: Vulnerability from system physical_db element vu_USB_port_DB

Include vu_coms_sniffing: Vulnerability from system app_layer element vu_coms_sniffing

Vulnerability vu_HD_not_encrypted "Hard Drive not encrypted ": IT: IT_GainInformation[
   attack "The attacker after stealing the Hard drive he can read the data in it."
   attackVector IT_Physical
   score 9.0
   description "The Hard Drive data is not encrypted"]

Risk th_hd_stolen "Hard drive Stolen": IT: IT_InformationDisclosure[
   vulnerabilities vu_HD_not_encrypted
   assessment [ probability 3 consequence 9 impact 3 impactLevel Low ]
   treatment t1: Mitigate [ solution g_hd_encrypt] ]
```

Spec. 2. GenericATM reusable system: RSL specification.

```
Package Projects.CashConnect_ATM

Import Catalogue4CyberSecurity.GenericATM.*

System CashConnect_ATM: Application: Application_Desktop [isConcrete vendor "Cash Connect" version "1.0"
   description "Automated Teller Machines (ATMs) distributed by Cash Connect"]

IncludeAll from system GenericATM

Goal g_privacy_rules"cash connect privacy rules": Privacy: Privacy_Disclosure[
   isPositive isAbstract isSolution
   description "ATM must comply with the privacy rules of the Cash Connect "]
```

Spec. 3. CashConnectATM concrete system: RSL specification.

6. Discussion

The Catalogue4CyberSecurity has been developed and assessed in different application domains. Table 3 summarizes the information captured and specified in RSL from the main references; this table is organized by the source that most contribute for the catalogue, including its packages and respective concerns. The current version of the Catalogue4CyberSecurity has considered major references from the cybersecurity community, namely the following sources: CVE details [4], Common Criteria [5], FIRST Common Vulnerability Scoring System [8], Beckers et al. [1], Core Security Patterns [21], and the Open Web Application Security Project [24].

The Common Criteria for Information Technology Security Evaluation (Common Criteria or just CC) [5] 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 and reliable, due to the fact these processes are already certified by the involved country members. Another approach used to develop the catalogue was to compile critical security vulnerabilities. The main reference used for that purpose was the vulnerabilities identified by the Open Web Application Security Project (OWASP) [24].

Table 3. Summary of the references considered by the Catalogue4CyberSecurity.

Source	**Asset Type**	**Packages**	**Risks**	**Vulnera -bilities**	**Requirements**	
					Goals	**Use Cases**
Common Criteria [5] (Organization)	Device (Various)	Various	-	-	10 (Positive, Concrete, Solution)	-
	Software Server (Various)	Various	-	-	30 (Positive, Concrete, Solution)	-
FIRST [9] (Organization)	Software Server (DBMS, Other)	DBMS	-	3	-	-
Beckers et al. [1] (Paper)	SoftwareServer (VM)	Cloud Systems	-	-	2 (Positive, Concrete, Solution)	-
OWASP [24] (Organization)	SoftwareServer (OS, DBMS, Other)	Various	31	9	-	6 (Negative, Concrete, Problem)
Core Security Patterns [21] (Book)	Application (Web)	Java Web Application	-	-	10 (Positive, Concrete, Solution)	-

OWASP is a not-for-profit organization that helps organizations to develop, purchase and maintain software applications that can be trusted. With the help of a variety of security experts from all over the world OWASP compiles every year a top ten list with the most critical web application security risks. Consequently, the vulnerabilities defined in the Catalogue4CyberSecurity include its 2017 list, which assures it is accurate and up-to-date. Finally, the book "Core Security Patterns" [21] provides a comprehensive set of patterns and pattern-driven approaches for the development of Web applications with Java programming language, including Web Tier, Business Tier, Web Services Tier and Identity Tier. These patterns were also classified in the RSL framework as positive, concrete and solution-oriented system goals for a general type of asset defined as Java-based web apps.

7. Conclusion

Cyber-security is a key concern in our society that can be analyzed from personal, organization or even government perspectives. Also, security shall consider different levels of systems or assets: from hardware and software infrastructures to business and organizational policies and practices, but also at application system level. However, security, regrettably, tends to be undervalued until a breach occurs and an organization suffers major damages. So the security of a system must be a concern as important as other concerns such as usability or performance, this way we can assure that our system is designed in a way to mitigate vulnerabilities and faults.

However, in spite of some approaches that have been proposed recently [10], [13], [16, 17], 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 risks, vulnerabilities and respective relationships. In addition, on top of this extended language we developed and created a catalogue of reusable security-specific specifications. This catalogue is in its starting phase, but shall be progressively developed with an increasing number of RSL packages by the community. Each package gathers the specifications that would be relevant for a particular 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.

For future work we plan to extend the RSL language with the possibility to include rigorous specification of security acceptance tests, for instance, based on the linguistic pattern "given-when-then" popular in BDD approaches [20,25]. Moreover, we intend to disseminate the Catalogue4CyberSecurity with its multiple packages for different application domains. We also intend to integrate it with other security frameworks and processes such as the SDL [14] to increase the efficiency of these processes with the aid of reusable requirements.

Acknowledgments

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

References

1. Beckers K., Côté I., and Goeke L.: A catalog of security requirements patterns for the domain of cloud computing systems, ACM Symposium on Applied Computing. ACM (2014)
2. Braeuer J., Gmeiner B., Sametinger J.: ATM Security – A Case Study of a Logical Risk Assessment, Conference on Software Engineering Advances (2015)
3. Caramujo, J., Silva, A. R.: Analyzing Privacy Policies based on a Privacy-Aware Profile: the Facebook and LinkedIn case studies, IEEE CBI, IEEE (2015)
4. CVE Details, The ultimate security vulnerability data source (2018), www.cvedetails.com. Accessed March 2018
5. Common Criteria (2018), www.commoncriteriaportal.org. Accessed March, 2018
6. Deng M., Wuyts K., Scandariato R., Preneel B., and Joosen W., A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements Requirements Engineering, Springer (2011)
7. Eclipse Xtext (2018), www.eclipse.org/Xtext, Accessed March 23, 2018
8. FIRST - Forum of Incident Response and Security Teams, Common Vulnerability Scoring System (2018), www.first.org/cvss. Accessed March 23, 2018
9. FIRST - Forum of Incident Response and Security Teams, www.first.org. Accessed March, 2018
10. Firesmith, D.: Specifying reusable security requirements. Journal of Object Technology (2004)
11. Ferreira, D., Silva, A. R.: RSLingo: An information extraction approach toward formal requirements specifications, MoDRE, IEEE CS (2012)
12. McDermott, J. and Fox, C.: Using abuse case models for security requirements analysis. Computer Security Applications Conference, IEEE CS (1999)
13. ITLingo - Specification Languages for the IT domain (2018), https://www.researchgate.net/project/ITLingo-Specification-Languages-for-the-IT-domain. Accessed March 2018
14. Microsoft, Security Development Cycle (2018), https://www.microsoft.com/en-us/SDL. Accessed March 23, 2018
15. Mouratidis H., and Giorgini P., Secure tropos: a security-oriented extension of the tropos methodology, International Journal of Software Engineering and Knowledge Engineering (2007).
16. Myagmar S., Lee A. J., and Yurcik W.: Threat Modeling as a Basis for Security Requirements, Symposium on requirements engineering for information security (2005)
17. Sindre G. and Opdahl A. L.: Eliciting security requirements with misuse cases. RE 10.1 (2005)
18. Silva, A. R.: Linguistic Patterns and Linguistic Styles for Requirements Specification (I): An Application Case with the Rigorous RSL/Business-Level Language, EuroPLOP, ACM (2017)
19. Silva, A. R., Caramujo, J., Monfared, S., Calado, P., Breaux, T.: Improving the Specification and Analysis of Privacy Policies: The RSLingo4Privacy Approach ICEIS (2016).
20. Silva, A. R., Paiva, A., Silva, V.: Towards a Test Specification Language for Information Systems: Focus on Data Entity and State Machine Tests. MODELSWARD (2018).
21. Stell, C.: Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management, Prentice Hall PTR (2005)
22. The Open Web Application Security Project (OWASP), https://www.owasp.org/index.php/Main_Page. Accessed March 23, 2018
23. The STRIDE Threat Model, msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx. Accessed March 23, 2018
24. OWASP, Web Application Firewall, www.owasp.org/index.php/Web_Application_Firewall. Accessed March 23, 2018
25. Wynne, M., Hellesoy, A., Tooke, S.: The cucumber book: behavior-driven development for testers and developers. Pragmatic Bookshelf (2017)