IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 20, 2021, accepted December 1, 2021, date of publication December 6, 2021,
date of current version December 24, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3133020

Reusable Security Requirements Repository Implementation Based on Application/System Components

FERDA ÖZDEMIR SÖNMEZ[1], (Member, IEEE), AND BANU GÜNEL KILIÇ[2], (Member, IEEE)
[1]Institute for Security Science and Technology, Imperial College London, South Kensington Campus, London SW7 2AZ, U.K.
[2]Informatics Institute, Middle East Technical University, Çankaya, 06800 Ankara, Turkey

Corresponding author: Ferda Özdemir Sönmez (f.ozdemir-sonmez@imperial.ac.uk)

ABSTRACT Forming high quality requirements has a direct impact on project success. Gathering security requirements could be challenging, since it demands a multidisciplinary approach and security expertise. Security requirements repository enables an effective alternative for addressing this challenge. The main objective of this paper is to present the design of a practical repository model for reusable security requirements, which is easy to use and understand for even non-security experts. The paper also portrays an approach and a software tool for using this model to determine subtle security requirements for improved coverage. Proposed repository consists of attributes determined by examining common security problems covered in state-of-the-art publications. A test repository was prepared using specification files and Common Criteria documents. The outcomes of applying the proposed model were compared with the sample requirement sets included in the state-of-the-art publications. The results reveal that in the absence of a security requirements repository, key security points can be missed. Repository improves the completeness of the security terms with reasonable effort.

INDEX TERMS Computer security, information security, requirement's engineering, software reusability.

I. INTRODUCTION

Requirements engineering is a process of defining, documenting, and maintaining requirements in the engineering design process. It is a common role in both systems and software engineering. If the documentation and storage of the requirements are done in significant amounts, then the structure or place used for this storage is called a repository. Security requirements due to their nature are repeatable from project to project, and thus, using security requirements repositories is common in the requirements engineering practices.

A software requirements specification (SRS) document is written to provide a path from a problem domain to a solution domain. In order to achieve this goal, it has multiple sets of requirements. An SRS document is utilized for formal correspondence among task partners, as well as for guidance in all the phases of the product creation. Writing a high-quality SRS is important, because it constitutes a base for the project. Gathering the proper requirements in the early phases of the project results in better designs and decreases the overall cost of the product development.

Writing good software requirements is not straightforward. As indicated by the IEEE 830-1998 [1] standard, "A requirement of quality should be correct, unambiguous, complete, consistent, ranked for importance and/or stability, verifiable, modifiable, and traceable". Composing security requirements is considerably more challenging. It requires a multidisciplinary approach based on requirements engineering, security engineering, and systems engineering. Most of the software teams (particularly in small organizations) do not include security experts. For this reason, quite often security is not handled properly in these projects.

Usually, detailed security analysis comes to the agenda too late, sometimes considered after the project is completed. In the best-case scenario, security issues are considered in the design phase without properly specified security requirements. This problem is more prominent in the web-based software development, because many of the web application development processes lack enough security support in the

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

early phases [2]. Because of these reasons, companies that develop and maintain software would benefit considerably if they could determine how to reuse security requirements from different projects naturally having similarities with each other. When software requirements are reused, not only the requirements, but also the related design, code, documentation, and tests can also be reused as they are or in a modified manner. This approach decreases the duration and costs of the projects. Companies that store their prewritten SRS know-how can use this knowledge for their incoming projects.

Using a security requirements repository which stores the security requirements in a structured way would help in not only gathering better security requirements, but also in other stages of the production. Defining assets of the project, proposing better alternative solutions to the problems and determining the constraints would become easier, and the overall result would lead to a higher level of security.

The aim of this study is complementing usual practices of requirements engineering, not replacing or changing them to reduce the difficulty of gathering security requirements. Requirements engineering includes processes, such as requirements elicitation, analysis, validation, and management. Users of the proposed model may continue applying requirements engineering practices which are found necessary or familiar. By applying the proposed methodology on top of existing requirements engineering practices, it is expected that reusability would increase.

The first and main contribution of this study is a repository model addressing the issue of reusing security requirements. It is expected that using this model would improve software security, eliminate the risk of omissions in the SRS document and increase the quality of the products. Reuse will also increase the effectiveness of IT companies and reduce the overall cost. Existing repository structures for reusable security requirements are reviewed in Section 2. The main advantage of this model over the existing models is its structure, which directly concentrates on the areas that are more prone to security problems. This model is designed to be filled in easily with known security requirement resources, such as SRS documents, standards having requirement lists, security handbook type books, and information coming from vendor sites and other online communities, and it is also expected to fit the requirements of all the stakeholders. The second contribution of this paper is a methodology for relating non-security requirements to security requirements that first utilizes keywords and then maps them to application components. This increases the usability and searchability in the repository. It is a known fact that majority of the existing reusable security requirements approaches lack automated tool support. The third contribution is a Python based repository search engine which fits to the proposed repository structure and allows an efficient search of the necessary requirements, given a textual description of a system. This study provides a model for storing the security data and a tool that runs with this model. Finally, two examples are provided to demonstrate the use of the repository including how to fill in the repository with data. However, preparing a real security repository is beyond the scope of this work.

The article has four main contributions: (i) a repository model addressing the issue of reusing security requirements; (ii) a methodology for relating non-security requirements to security requirements that first utilizes keywords and then maps them to application components; (iii) a Python-based repository search engine which fits to the proposed repository structure and allows an efficient search of the necessary requirements, given a textual description of a system; and (iv) two examples to demonstrate the use of the repository including how to fill in the repository with data.

The remainder of the paper is organized as follows: Section 2 has the literature review. Section 3 has the methodology description. Section 4 introduces the proposed security model. Section 5 explains the empirical study design with two case studies and a validation workshop including the presentation of the proposed methods and earlier methods for comparison, application of the proposed method and a user experience survey. Section 6 presents the results of using a sample repository for two different cases and workshop results. Section 7 is a discussion of the proposed study. Finally, Section 8 contains the conclusions and further research possibilities.

II. LITERATURE REVIEW

Frequently, security becomes a subsection of the researches related to the application or system development [3]. However, since the development of security requirements is critical, several frameworks and models have been proposed each having specific approaches. The first group of requirement engineering methods focusing on security provide ways to find security requirements without saving knowledge. They are methodological in general and designed to be used mostly during the requirement elicitation and analysis phases. The second group of requirement engineering methods focusing on security may or may not provide methodologies, but they serve as a model to save a set of security knowledge for later use.

Studies focusing on security requirements reusability selected different features to be reused. These features include requirement statements, security patterns, security goals, countermeasures, threats, attacks, assets, organizations, and vulnerabilities [4]. The representation of the reusable features also differs. In general, catalog/taxonomy-based approaches store textual statements. Non-textual representations include UML usage, ontology usage, or specifically defined sets of shapes to define patterns/scenarios. The categorization of the reusable security knowledge is made based on assets, threat or attack types, security objectives, security properties, countermeasures, and security tests. Reusable textual statements include actual requirement statements, generic or prototype requirement statements, textual attack scenarios/patterns, attacker goals (anti-goal), and misuse case scenarios. Graphical representations may be formed of graphics scenario definitions including actors, such as users, systems administrators, and attackers, assets, actions, and states.

The proposed model is based on saving textual security requirements in a repository for reuse. Although the UML class diagram is used to describe the proposed model, the model does not store UML diagrams, but textual requirement statements in a classified manner. The categorization criteria for the requirement statements are based on application components/features which correspond to a mixture of assets, architectural properties, infrastructure elements, and application functionalities. In order to get more information related to studies that take security goals, countermeasures, threats, attacks, assets, organizations, and vulnerabilities as reusability items, please refer to Souag *et al.* [4].

One of the well-known security resources is Common Criteria (CC). CC is a framework for information technology security evaluation, which is accepted as an international standard and abbreviated as ISO/IEC 15408 [5]. The main purpose of the CC framework is to provide a platform for comparison among independent security evaluations. For this purpose, it provides a process model that works on sets of security requirement templates. More information about the CC framework can be found on the Common Criteria portal website [6]. The most significant similarity between the CC and a security requirements repository is having existing sets of defined requirements templates.

In CC the security requirements templates are given in two different documents; functional security requirements [7] and assurance security requirements [8]. The requirement sets in CC are very generic and template-based. They have to be tailored and worked on to be used in an actual project. CC offers a tailoring process that includes iteration, assignment, selection, and refinement. Tailoring the sets of security requirements templates would require a security expert view so as to accomplish a decent set of security requirements at the end. CC model does not take advantage of existing SRS documents. It does not offer an approach to relate non-security requirements to security requirements, either. Mellado *et al.* [9] offered a process for requirements reuse by integrating CC usage into software development practices. Saeki and Kaiya [10] proposed a requirements elicitation method based on CC. In Mellado *et al.*'s study, the objectives that fit the solution space for the specific project/case are taken from the CC catalog, furthermore, CC security function definitions are benefited to form the security requirements.

MAGERIT is another framework for risk analysis and management [11], that is developed by the Spanish Ministry of Public Administrations. It enables risk analysis and prescribes countermeasures for each risk. These countermeasures are counted as security requirements. Its main focus is analyzing the requirements, however, reusing requirements is not targeted. The MAGERIT asset hierarchy was exploited by the SIREN (Simple Reuse of Software Requirements) method in its repository model [12] and ontology-based study by Lasheras *et al.* [13].

SIREN attempts to translate the ''security measures stated in MAGERIT into reusable security requirements'' [11]. Its approach is based on creating a document hierarchy and document templates. The SIREN model stores reusable requirements using domains and profiles which correspond to a homogenous set of requirements that can be applied to a domain. The SIREN model has a particular labeling system for requirement trees. Its main drawbacks compared to the proposed model are that it does not relate non-security requirements to security requirements, and even though the document structures have their labeling systems that give information on the classification of the requirement, they do not give enough information on the content of the requirement, which would eventually make it difficult to search in the repository and cause usability problems. Nevertheless, this approach was used along with an audit method for a healthcare application [14]. The results of the audit were reported as an audit report.

Firesmith's [15] model is also a study that stores security requirements as reusable knowledge. In this model, valuable assets, their identification, threats to these assets, and negative impacts are identified. Filling this repository model requires relatively high resource and time usage to construct a dependency relationship among a large set of security-related objects including security goal, security policy, security requirement, security mechanism, security risk, security quality factor, security mechanism, security risk, threat, attack, attacker, vulnerability, harm, system, asset, property, people, service, data, hardware, software, facility. This repository model allows the storage of a comprehensive set of knowledge, however, for some IT companies which prefer practicality over the availability of all security-related data, it may not be the first solution.

Using the use cases to define security requirements is another approach. There have been several studies proposing the use of use-case modeling techniques, such as UML, to define security requirements [16]. In this approach, once the valuable assets are identified, the misuse scenarios of the system are defined using use-cases, forming the threat scenarios. Security scenario(s) related to each misuse scenario are also defined by the same approach. These security scenarios include bundles of security requirements, which may be reused for similar use-cases again and again.

Zuccato *et al.* [17] presented a process that is based on not reusing security requirements, but security requirement profiles, which is close. They proposed a process that included risk analysis, questionnaire, selection of requirements, and grouping requirements. These requirement groups are forwarded to suppliers. This method provides requirement groups for different business profiles. This method has similarities to the proposed method such that groups of application components which will be explained in Section 4 may also correspond to business profiles and technological divisions.

Lasheras *et al.* presented an ontology-based security requirements repository [13]. They implemented a model using Ontology Web Language, (OWL), which is recommended as an information-sharing mechanism by W3C. This is a lightweight model permitting the detection of inconsistencies and incompleteness of security requirements and allowing the reuse of security requirements. This approach requires learning a special language, OWL, and tools for the implementation of the ontology and consistency checking. When building and using requirement ontologies, having domain-specific ontology sets and corresponding tools, instead of generic OWL tools results in the higher usability of the systems [18]. There are other ontology-based earlier security requirements studies. For example, Salinesi *et al.* [19] proposed a method for the requirements elicitation based on ontologies.

Model-driven engineering approaches also commonly depend on the repository usage. Hamid [21] used models to ease, systematize and standardize the software development process, not only the requirements phase. The repository stored domain-specific modeling artifacts. The repository elements were characterized based on the keywords, lifecycle stage search, and relationship types. As modeling tools, UML, profiling, and Ecore modeling language were used. Mainly this repository held a dictionary of binaries, languages, and artifacts in compartment forms. Unlike security requirement repositories, model-driven repositories focus on the whole project lifecycle components. In another work, Hamid and Perez [22] provided a repository model to reuse the features (techniques, measures, etc.) related to domain standards specific to the engineering of embedded systems with safety requirements.

Although the authors could not find any solid security requirements repository model approach based on security patterns [23], the strong relationship of security patterns with the reusability and existence of various security patterns developed by a number of earlier studies makes it necessary to mention the security patterns in this paper before continuing further. A security pattern describes a recurring security problem and points out a generic solution for it. While doing this, the context of the problem can be described by using an example problem or a situation. Patterns can be used for secure development, testing as well as defining the security problems.

Earlier studies had two main goals in general. The first goal is to allow formal ways for the definition of reusable knowledge (action, scenario, goal, use-case, policy) and the second goal is to improve requirements reuse. For the first goal, either textual or graphical languages were defined (new language definitions) or already existing ones were used (such as OWL) as pattern characterization languages. For the second goal, each individual study provided a number of patterns that may be specialized in time by including other behaviors and actions. KAoS [24] is a tool-based policy services framework that is also the most mature ontology-based security pattern development approach. Other examples of pattern-based studies include Hermoye *et al.* [20] which defined attack patterns (anti goals) using real-time linear temporal logic expressions and used replay attacks to demonstrate. Alrajeh *et al.* [25] points out the importance of identification of the attributes related to software's environment during the definition of software requirements. Supaporn *et al.* [26] used security patterns to construct a grammar of security requirements. In this way, the authors expected to define complete and correct requirements. They used the authorization security pattern to demonstrate the grammar usage. Another pattern-based security representation is Secure Tropos [27]. Tropos [28] is an agent-oriented software development methodology based on the clear identification of system actors, resources, goals, and their dependencies. This methodology is extended by including four new concepts to the agent-oriented system definition mechanism, "Security Feature", "Protection Objective", "Security Mechanism", and "Threat". The existing methodologies described in this section before security patterns do not provide a comprehensive list of categorization items for a security requirements repository. Riaz and Williams [29] claimed that security patterns can fill this gap by providing groups of security patterns focusing on groups of specific issues.

Some interesting research has been carried out in this area in recent years. Mazo and Feltus [30] proposed a conceptual model which relies on dividing security requirements into simple security patterns. They suggest the use of set theory later to obtain more complex requirements. This has a similarity with the current proposal in terms of using simple pieces to form complex requirements. The drawback of this study there is no validation of the ideas presented and no demonstration of how the security requirements will be composed of simpler parts. Gonçalves and Silva [31], [32] provided a design that relies on a more robust framework, RSLingo [33]. They extended the RSLingo requirements specification language to define security requirements. Forming high-quality requirement statements is directly related to the use of natural language properly. Silva and Savić [34] examined the use of linguistic patterns to define security requirements which resulted in better requirements specifications written more systematically and consistently. One recent advancement on the use of security patterns for reusability is by Salva and Regainia [35] in which data integration was used to express software attacks, security principles, and security patterns. This has a larger scope compared to other security pattern studies. Ramadan *et al.* [36] proposed the use of a BPMN-based framework extension along with a semi-automated process to express security requirements. This allows a graphical representation of security issues along with functional requirements. Another recent approach is the use of formal methods to verify compositions of security patterns for Scada systems by Obeid and Dhaussy [37]. An effort is made by Wirtz and Heisel [38] to bridge the gap between the functional requirements and security requirements which provided a template with a form structure to store the relations of functional and security requirements. Although the model looks useful when used as a part of a single project, it does

TABLE 1. Strengths and weaknesses of the selected studies.

Common Criteria Model [6]	• Accepted as an international standard. • Provides a platform for comparison.	• Requirements are very generic. • Requirements have to be tailored and worked on to be used in a real project • The repository size does not grow. • Does not take advantage of existing SRS files. • Does not relate non-security requirements to security requirements.
Siren Model [12]	• Taking advantage of MAGERIT's risk analysis, risk management methodologies together with its asset hierarchy model. • Creates an independent document hierarchy model	• Requires learning MAGERIT asset hierarchy model • Does not relate non-security requirements to security requirements. • Carrying out search in this model looks hard which decreases usability • Created document hierarchy model does neither take advantage of own existing SRS documents, nor be associated with them
SEI Model [15]	• The model depends on good theoretical knowledge on security issues and thus includes a large set of object types in itself.	• Difficulty arises while constructing the connections between these large sets of different object types • This makes it impractical and infeasible to be used by IT companies. • Implementing the model is not easy. • Does not offer methods for searchability.
UML Based Approaches [16]	• If the organization has already adopted UML based techniques as a part of requirement analysis practices, utilization of UML based approaches would be easy and appropriate. • Might be used for some complex security requirements that include multiple steps of actions, regardless of UML usage, as a supplementary to other repository approaches.	• Using this technique for all security requirements is very time consuming resulting little practicality.
OWL Based Approaches [13]	• OWL is suggested as a legitimate data sharing mechanism and this is a leverage of ontology-based repository models.	• Using OWL based repositories requires utilization of specific tools for data entry and data exploration. • It also requires the use of specific tools to check the consistency of the information stored. • Unlike UML, in authors' knowledge, the use of OWL language and tools are not common for requirements engineers.
Security Pattern Based Approaches [20]	• Similar to UML based approaches, these models have varying levels of benefits. • Security patterns may be used to demonstrate some selected requirements.	• The authors did not encounter any security pattern that suits or that is used to model a large set of requirements. • Even if the security pattern is suitable to represent a large set of security requirements, this approach may not be as efficient as using textual representations. Actually, existing security pattern studies only demonstrate a single or a few security requirements at most.
Proposed Model	• Easily takes advantage of SRS files and other listings • Grows easily by adding new projects data • Having a tool support causes high usability and increased searchability via keyword and component names. • Requires a short learning period. • May allow creation of a shared repository platform easily • Relates non-security requirements to security requirements and in this way decreases the possibility of omitting necessary requirements	• Being a new system • Requirements do not have a generic format. Thus, some simple wording change would be necessary when passing the requirement sentence to the formal security requirements set. • Does not include risk factors, risk analysis and management in the repository model.

not provide evidence on the reusability of the stored security requirements across multiple projects.

The scope of literature review section is limited to earlier studies which provide a model to reuse security requirements information. Some related concepts such as integration of requirements engineering methodologies with the overall lifecycle is left out of the scope of this study. Existing similar studies are elicited based on reusable knowledge. Although, there may be some secondary reusable information such as assets, risks, threats, attack types or vulnerabilities in a few of the presented studies, the focus of this paper is the studies which have the main reusable information as textual reusable security requirements or as textual reusable requirement templates. Thus, other studies which enable repetition of security related tasks among multiple projects such as a generic risk assessment methodology, but that do not focus on the reuse of textual security requirement statements are left out of the scope of this study.

The authors made a summary of the strengths and weaknesses of the proposed method and selected earlier methods described in the literature section in Table 1. One important common weakness is the lack of automated support for the majority of the earlier approaches. This issue was also highlighted in the survey study by Souag *et al.* [4]. Souag *et al.*

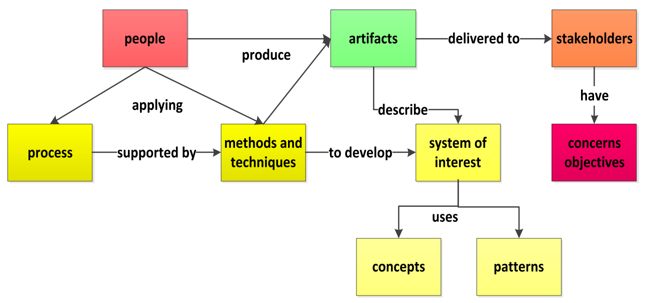


FIGURE 1. Context diagram used to find concepts associated with reusability of security requirements (adapted from [39]).

examined 95 studies that are related to the requirement phase and contribute to the decision of security requirements in various ways. Only the 13.6% of these studies had automated support.

The earlier methods mentioned in this paper, also, lack, in general, the information where the security requirements should be used, i.e., they define what should be done to some extent, but they do not specify enough when and how this information should be used. Riaz and Williams [29] also claimed that security patterns are superior to the repository models, in terms of characterizing the context, where and when the security requirement should be used. The proposed model may have some correspondence regarding these benefits of security patterns which will be discussed in later sections.

III. METHODOLOGY

The study started with two questions, Fig. 2. The first one is

- "Which structures aim or are more beneficial for storing reusable security requirements?".
- The second one is
- "What are the sources of requirements to fill these structures?".
- After these initial questions, two secondary questions turned out.
- "Are these existing structures adequate?"
- and
- "Is there a better and practical way of storing this kind of data?"

With the guidance of these questions, the study involved identifying the structures for storing reusable security requirements and searching for the sources of requirements to fill these structures. The study also included evaluating the adequacy of existing structures and exploring better and practical ways of storing this kind of data. In order to take these steps, first, a detailed literature survey was carried out and existing studies that are suitable or related to the presented purpose were examined. After this examination, the advantages and disadvantages of these methods were noted.

Having these research questions, literature review results, examination results for data sources, and the domain knowledge, the hypothesis "If a repository structure which is naturally congruent with the security requirement sources is

TABLE 2. Research concepts.

People	Security Requirements Engineers, System Engineers, Software Engineers
Process	SRS preparation (requirements engineering processes)
Artifact	Security requirements statements
Methods and Techniques	Create new security requirements, Reuse security requirements as is, Reuse security requirement templates
System of Interest	Standards that include security requirements, Structures that allow restoring requirements statements or templates for reuse
Stakeholders	Software Development Companies, Vendors Companies, Standard and Regulation Owners, Security Evaluator Companies, Security Analysts, Project Manager, System Admin, Security Auditor, Customer
Concerns Objectives	Available Security Requirement Sources, Evaluation & Certification Standards, Obeying Standards, Regulations, Obeying Regulations, Business Requirements, Handling Security Problems Caused by Business Requirements, System Requirements, Handling Systems Security Requirements, Network Infrastructure, Handling Network Security Requirements, Software Requirements, Handling Software Security Requirements, Project Management, Practicality of System, Reusability of System, Correctness of System, Accessibility of System
Concepts	Repository, Ontology, Taxonomy, Spreadsheet Design, Standard, Guideline, Database, Functional Requirements, Non-Functional Requirements
Patterns	Repeated Threats, Repeated Vulnerabilities, Repeated Other Issues (Can be classified using tags)
	Repeated Behaviors and Requirements of Actors, Repeated Functionalities

created, using this structure may be expedient for the companies having both their existing security requirement know-how and available public sources" is emerged.

Since forming the aimed structure needed a system engineering type process, a system engineering concept diagram, as shown in Fig. 1, describing the decisions would be appropriate. The concepts associated with concepts found in Fig. 1 are provided in Table 2. These concepts helped to form the final structure of the proposed design. The majority of the research concepts presented in this table are either converted to application/system features to cover the stakeholder requirements or caused a design and/or implementation decision. The information in this table will also be used to describe the design decisions more systematically in this section.

The study is started knowing the group of "people" who needed and who are responsible to create the main "artifact" "Security Requirement Statements" during the main

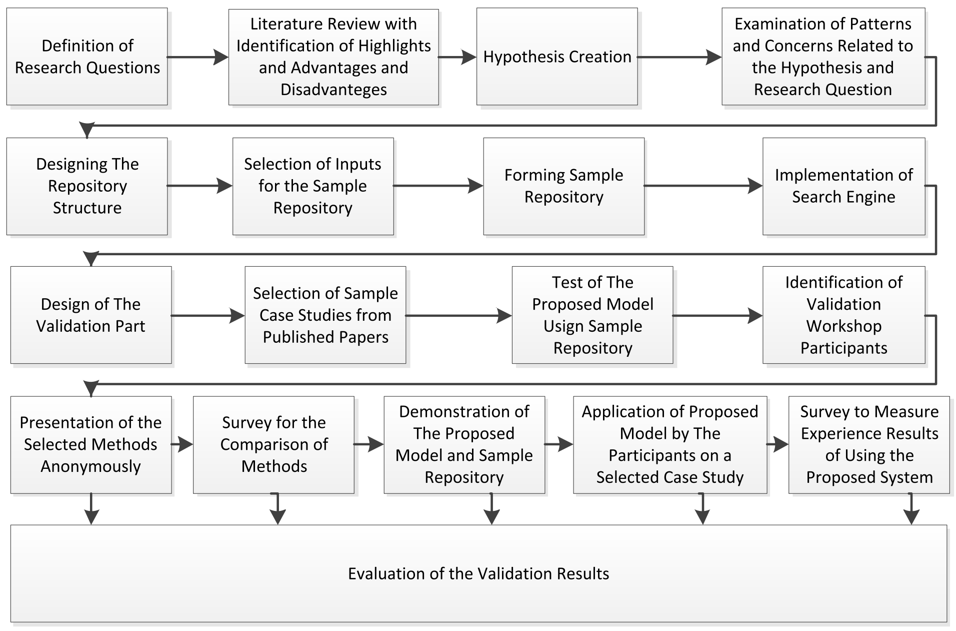

Definition of Research Questions
Literature Review with Identification of Highlights and Advantages and Disadvanteges
Hypothesis Creation
Examination of Patterns and Concerns Related to the Hypothesis and Research Question
Designing The Repository Structure
Selection of Inputs for the Sample Repository
Forming Sample Repository
Implementation of Search Engine
Design of The Validation Part
Selection of Sample Case Studies from Published Papers
Test of The Proposed Model Usign Sample Repository
Identification of Validation Workshop Participants
Presentation of the Selected Methods Anonymously
Survey for the Comparison of Methods
Demonstration of The Proposed Model and Sample Repository
Application of Proposed Model by The Participants on a Selected Case Study
Survey to Measure Experience Results of Using the Proposed System
Evaluation of the Validation Results

FIGURE 2. **Methodology summary.**

"process", SRS Preparation as a part of Requirements Engineering tasks in advance. Since this study aims to provide a "method/technique" for reusing the security requirements, two decisions are made after the literature survey in which the "systems of interest" are examined in detail, the manner of reuse and the form of reuse. The manner of reuse is determined as reuse of "security requirements as is" or "in a modified manner" and the form of reuse as "textual security requirements" and "textual security requirement templates". Text based requirements have their own advantages and disadvantages. First of all, text-based requirements allow the use of many requirement resources, such as books, best practice documents, standards and earlier SRS documents. Requirement repositories based on expression ways other than natural languages require comparatively more time to prepare.

The main "concerns" related to the decisions of the form were the availability of the security requirement sources in this form and the practicality of using the reusable information in this manner.

One important factor which affected the tasks during the study is that, the majority of the existing methods lacked providing ways to collect the corresponding requirements data, despite including a description of the structure or a process to store reusable security requirements. For this reason, the authors searched sources of security requirement sets to improve domain knowledge. Some requirement sets existed in small amounts on some other available documents, such as SRS files shared on the Internet, and scientific articles, however, the standards, the vendor sites, and the security forums discussing security requirements related to a particular application, architecture, or technology consisted of large sets of available input for a reusable security requirements repository.

The "concepts" and "patterns" that are related and are beneficial for implementation are examined, in parallel to the identification of "stakeholders" and their needs, concerns, and priorities as shown in Table 2. The concepts associated with the stakeholder requirements, and the concerns, and patterns found out during this examination yielded the elements of the proposed structure. Specifically, some of the stakeholder concerns such as "network requirements", "software requirements", "standards", and "regulations" corresponded to some elements in the resulting design. The tasks of this study included the design of the proposed model, the establishment of a data structure, implementation of the search engine tool, using content analysis technologies to find out keywords. This is followed by an empirical study design for validation purposes. The empirical study designed for validation has two parts. In the first part, to examine the results of using a security repository, a sample repository library was prepared that is used for two case studies from two published articles. The aim was not to prepare a complete repository with complete sets of keywords. However, even having a sample repository was sufficient to demonstrate the usage and the benefits of the model. In the second part, a workshop was designed to enable the evaluation of the proposed model by users from the industry. During this workshop, a demonstration of the proposed model and all the mentioned previous models had been made. Following this, a survey was conducted to enable a comparison of these methods. The workshop also included an experimental use of the sample repository for a selected use-case. The experiences of the workshop attendees were collected and evaluated at the end of the validation section. The details of this empirical study design are explained in Section 5, after the description of the proposed model.

The technical decisions of the study were also made gradually. Initially, the storage media was decided to be a database rather than an ontology or a taxonomy document for its wide acceptance and ease of use. However, during the implementation phase, due to the power of the fourth-generation languages, Python came forward for this case. For working with spreadsheet data, the initial structure was normalized to fit in a spreadsheet design without changing the process methodology and model elements.

IV. PROPOSED SECURITY SYSTEM

Security experts recommend mechanisms to satisfy security targets, confidentiality, availability, integrity, and non-repudiation, of a system, such as authentication and authorization mechanisms for accessing an asset to ensure confidentiality or replication and backups to ensure availability. Threat analysis provide security risks and this effort ends up with adding special functional or non-functional security requirements to the system specification. Traditional risk analysis techniques which include analysis of security threats, identifying assets and defining security measures are necessary to provide an overall secure structure for an application. However, this traditional approach stays at an upper level during the requirements analysis and may not be capable of detecting some lower-level security requirements. For example, an application architecture that includes cookie usage should have a statement in its specification, like "Careful design of cookie contents and removal of sensitive information are needed because cookies are transferred in plain text". An application which depends on stateful user sessions should have a statement like "Application sessions must be closed automatically after an inactivity period.". If a statement like "Linux/Unix servers will be used as web servers." is included in the requirement statements, then it is good practice to include a statement like "Login shells on Linux/Unix machines should be disabled during the deployment of the system". Hundreds of examples like these can be listed. In fact, almost all application/component features have their own particular, negligible threats or vulnerabilities, which can only be prevented by systematically catching these application/system components, such as depending on keyword searches.

Software is an ill structured problem, i.e., the problem and the solution cannot be separated from each other causing another practical issue. Requirements frequently change or evolve during development, even later in the development phase. Therefore, a thorough understanding of the system under development, which is necessary for the risk analysis, may not always be possible for the security expert. Even a minor change of an existing functionality or a newly added function may lead to a significant vulnerability, which cannot be easily detected among all the other details of the project. Therefore, there should be an approach for automatically determining security requirements when a functional or non-functional requirement changes or a new functional requirement is added, thereby the coevolution of non-security and security requirements can be achieved.

The third practical problem is identifying security requirements in a large amount of text often used in specifying requirements. Vulnerabilities may be hidden among pages and pages of text and even a large group of accomplished security experts may miss subtle requirements. This issue is similar to code inspection. Even an expert would miss problematic pieces of code during a visual inspection. Hence automated tools are used, yet results still demand human touch and interpretation. Similarly, the security requirements repository proposed in this work aims to make it easier to determine and reuse security requirements and ensure comprehensive coverage. It does not aim to replace traditional security risk analysis; it aims to support that. As a result, the amount of effort and time required for the project development can be reduced, while the security is improved.

In the proposed security model, applications or systems are divided into components or features as a rule. These components or features do not refer to application modules. In this article, it is used to define everything that affects the application or system security. For example, when a new type of actor starts using an application, it may cause new vulnerabilities due to the use of different functionality or different access rights. If the application runs on a particular operating system, this may cause some vulnerabilities. If the application has to obey some kind of a regulation, this may bring in additional kinds of security issues. Various application components and features are characterized in this model.

The list of components and features, which will be called as class types from now on, are: A-Actor, B-Application Architecture, C-Authentication Mechanisms, D-Access Control Mechanisms, E-Browser, F-Data, G-Database, H-Functionality, I-Hardware, J-Language, K-Middleware L-Networks, M-Operating Systems, N-Payment Systems, O-Physical Protection, P-Protocol, Q-Regulation, R-Stakeholders, S-User Interface Devices, and T-Tags.

Each class type is selected with caution due to relations to vulnerabilities and threats. It is known that different actors (A-Actor) may require different security needs or may cause specific vulnerabilities or threats. Application architectures (B-Application Architecture) also have inherent security requirements. For example, a mobile application is different from a cloud application, or a web-based application is different from a desktop application. While protecting the authenticity of the users, each authentication mechanism (C-Authentication Mechanism) may bring additional security requirements. For example, a password-based authentication system will require an addition of password constraints definition or automatic control of password strengths. Similar to authentication systems, access control mechanisms (D-Access Control Mechanism) may have their own requirements, such as a definition of ownership of assets. Browsers

(E-Browser) have security flaws. Some systems re-quire the use of specific browsers or are designed to be used on specific versions of the browsers. Knowing the browser types may result in additional protections specific to the vulnerabilities of those browsers. Data (F-Data) has its own security requirements. For example, personal information, patient health information, government secrets would cause data specific security requirements. Similar to the browsers, databases (G-Database) have their own kind of vulnerabilities, such as SQL injections and backup storage problems. Specific functionality (H-Functionality) would require specific care of security. During the implementation of functions such as file upload and download, search functions, or privileged user functions, using security requirements which obey best practice rules would be very beneficial. Hardware (I- Hardware) may also be the source of some vulnerabilities. Vendor sites publish such vulnerabilities periodically, so such information may be saved in the repository for later use. Some development languages (J-Language) are more prone to security failures than others. For example, script type languages have known vulnerabilities; other languages such as Java, PHP have also known vulnerabilities which have to be dealt with specifically. Middleware (K-Middleware) may cause problems, such as deployment failures, replication failures or load balancing failures. Network (L-Network) originated vulnerabilities would form the largest group among others. Although operating systems (M-Operating System) continuously update themselves to improve security, each still has some specific vulnerabilities which should be dealt with carefully. This kind of vulnerabilities can be found in Common Vulnerabilities and Exposures database, benchmark sites and operating system vendor publications. Among all other functionalities, payment systems (N-Payment System) are given more importance. They are different than other functionalities, as they may require the use of specific devices, regulations, network transmission protocols or a combination of them and they are more intriguing to people with malicious intents. Places which require physical protection (O-Physical Protection) would require corresponding security checks, such as sensor checks and CCTV camera configurations. There are many protocols (P-Protocol) which have specific security requirements, such as cryptographic protocols. Working on such protocols and defining the requirements is a time-consuming process and once it is done such information may be stored in the repository to be used in similar projects later. Sometimes, software development agreements force regulations (Q-Regulation) and regulations force some security requirements. Examples of such protocols include Computer Fraud and Abuse Act (CFAA), Foreign Intelligence Surveillance Act (FISA) and Health Insurance Portability and Accountability Act (HIPAA). Similar to protocols, determining requirements for these regulations is also very time consuming and saving them would result in using less time and more complete security requirement sets. Stakeholders (R-Stakeholders) may cause new security requirements, such as integration and verification requirements. Finally, some user interface devices (S-User Interface Devices) such as keyboards, handheld devices, and smartphones have specific security requirements. Besides the other categories found in the repository, it will be valuable to attach some labels to the application or system, because some concepts are hard to characterize as an application or system component. "Requires Cryptology", "Uses Audit Mechanism" are examples of tags (T-Tag) defined so far in this study. These tags refer to particular types of security requirements for a project. Selection of these features or components has been made in the wake of examining common security problems. They include most of the common sources of security problems covered in security books, articles, and web indexed lists. These component sets can be extended by the users of the repository model by including new class types as the technology evolves.

Reusable security frameworks mostly rely on finding requirements and preserving the relations between them. In the proposed model, security requirements and non-security requirements are segregated, since the security requirements are usually specified after non-security requirements. Unless non-security requirements have been defined, the effort for defining security requirements cannot bring about concrete results. Therefore, the first step of using this repository is to define non-security requirements. Non-security requirements are isolated into two subsections; non-security non-functional requirements and non-security functional requirements. Whenever new requirements are included in the requirements document (SRS), the repository can be queried multiple times for new security requirements. The distinction of functional and non-functional requirements has been made in both security and non-security requirement parts on purpose to impel the user of the repository to proper sources. Requirements engineers or system analysts perform the requirement engineering steps so as to accumulate all the non-security requirements. This article does not deal with the details of the requirements engineering processes, whose methodology can be tailored according to the project.

The sample queries listed below demonstrate how security definition sources can be used to fill in a repository and how the stored information can be queried based on application/system features or corresponding keywords which may appear in non-security requirement definitions.

- *Security requirements related to some specific vulnerabilities/threats*
- *Security requirements related to databases or a specific database type, such as Oracle or DB2*
- *Security requirements which were defined in the last year or from a specific reference*
- *Security requirements related to system administrators/cloud user*
- *Security requirements related to ATM payments, file transfer or password reset functionalities*
- *Security requirements related to ATM user interface devices or keyboards*

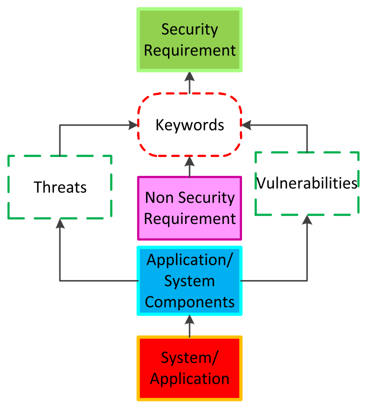


FIGURE 3. **Top level relationships of the repository model elements.**

• *Security requirements specific to PHP development language*

• *Security requirements assigned "Token" or "Default Passwords" tags*

• *Security requirements related to HIPAA standard or TCP-IP protocol*

• *What are the non-functional security requirements (which may possibly correspond to managerial decisions)*

• *What are the functional security requirements (which may possibly correspond to technical solutions).*

If the strengths and weaknesses of the proposed model are further examined, Common Criteria appears powerful due to international acceptance as a standard and providing an evaluation platform. However, its requirement sets are very generic and have to be tailored and worked on prior to actual use in a real project. In addition, to include support for the security requirements of new technologies, consensus of broader organizations is required. While determining requirement sets, CC does not take advantage of existing SRS files and does not relate non-security requirements to security requirements.

A. REPOSITORY TERMINOLOGY IN RELATION TO COMMON CRITERIA

Before going further on the repository model, the authors decided that using some of the Common Criteria (CC) terms to explain the approach of this article will make it easier to understand it, for the type of audience who has experience and/or knowledge on CC. CC uses the term component to describe the smallest selectable set of elements on which requirements may be based. This definition is very parallel to the term application/system component/feature term used in this paper. Another important term used in CC is the target of evaluation (TOE) term. TOE is the combination of sets of software, firmware, hardware and environmental devices which are subject to security evaluation. In CC, TOE may be in diverse forms, like a document or an installed software. In this approach, it is required that at least TOE specifications should be in a document format to further work on it. It corresponds to the system/application term in this model.

CC separates the sufficiency checks and correctness checks for a TOE. In this repository approach, existence of the necessary requirement statement is the target of evaluation. This requirement statement may be related to the sufficiency of a countermeasure or a correctness or both. If the necessary statements are found in the repository, then this would result in both correct and sufficient countermeasures for a vulnerability.

CC uses the terms packages and protection profiles for the sets of security requirements. Packages may contain security functional requirements, SFR's or security assurance requirements, SAR's. These packages are intended to be reusable. In fact, the information that are stored in the repository are the security functional requirements and security assurance requirements. Protection profiles refer to the requirements which are related not to a TOE, but to a class of a TOE, such as firewalls, smartcards, etc. These protection profiles are typically written by user communities seeking consensus for a requirement, developer of the TOE or a government or large organization as part of an audit process.

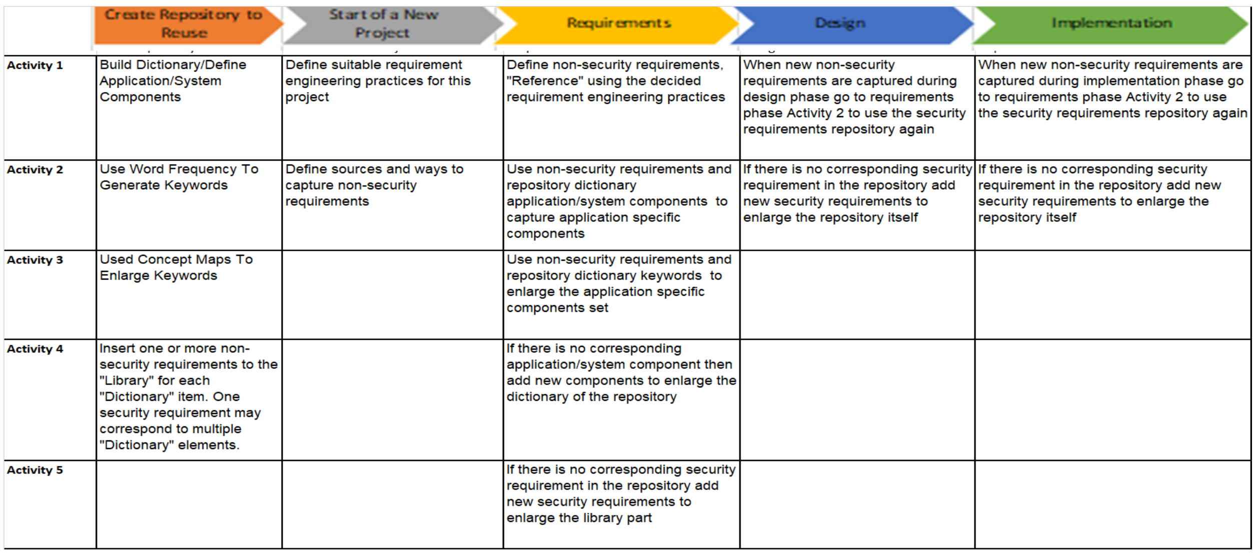
At the first look, this model may seem like lacking a protection profile classification to identify the already classified requirements. In this model, the requirements were taken out of packages or protection profiles and stored in the repository. They were indexed using lower-level terms, components. Indexing these requirement sets in this way increases reusability. Moreover, the tag field can also be used to keep the protection profile information.

Security target, which is the last term taken from the CC, is used to describe the security objectives for the TOE and its operational environment. In order to decide on the security target, the security problems should be identified, security objectives should be decided on and so on. This repository model does not include any such process to identify necessary objectives, protection profiles and the resulting security target for a system/application. These requirement engineering practices are excluded to avoid forcing the user to a particular methodology or process. As mentioned earlier, the authors recommend that the users of this repository should continue using their existing requirement engineering practices in parallel to the repository during the identification of the security objectives for a system/application.

In this model, the TOE or system/application is identified together with its related non-TOE software, hardware or firmware. This is also recommended by the CC. Application/system components in the proposed model can have networking elements, hardware elements, elements related to other software classes, etc. Without knowing its environment, necessary security countermeasures can not be installed for a system/application.

B. EXPLANATION OF THE REPOSITORY USING A CLASS DIAGRAM

The class diagram of the model can be seen in Fig. 9. Non-security requirements form "Reference" part of the repository, which is shown in pink color in the figure. These



	Create Repository to Reuse	Start of a New Project	Requirements	Design	Implementation
Activity 1	Build Dictionary/Define Application/System Components	Define suitable requirement engineering practices for this project	Define non-security requirements, "Reference" using the decided requirement engineering practices	When new non-security requirements are captured during design phase go to requirements phase Activity 2 to use the security requirements repository again	When new non-security requirements are captured during implementation phase go to requirements phase Activity 2 to use the security requirements repository again
Activity 2	Use Word Frequency To Generate Keywords	Define sources and ways to capture non-security requirements	Use non-security requirements and repository dictionary application/system components to capture application specific components	If there is no corresponding security requirement in the repository add new security requirements to enlarge the repository itself	If there is no corresponding security requirement in the repository add new security requirements to enlarge the repository itself
Activity 3	Used Concept Maps To Enlarge Keywords		Use non-security requirements and repository dictionary keywords to enlarge the application specific components set		
Activity 4	Insert one or more non-security requirements to the "Library" for each "Dictionary" item. One security requirement may correspond to multiple "Dictionary" elements.		If there is no corresponding application/system component then add new components to enlarge the dictionary of the repository		
Activity 5			If there is no corresponding security requirement in the repository add new security requirements to enlarge the library part		

FIGURE 4. **Timeline of proposed repository related activities.**

non-security requirements should be the basis to capture the security requirements. To use the knowledge stored in the Reference part, the "Dictionary" and "Library" parts of the repository should be filled with data. Each non-security requirement may be the reason of the presence for one or more application/system component/feature in the selected set. These relations of non-security requirements to application/system components/features are captured by searching the application/system component/feature name from the requirements document and as a second step searching for application/system component/feature specific keywords in the requirement documents to capture more subtle application/system component definitions. The definition of "Reference Part" requires the start of a new project, so definition of "Dictionary" and "Library" parts related data can be earlier in time. However, unless "Reference" data is defined, nobody can actually start benefiting from the repository. A timeline of these activities is provided in Fig. 4. In Table 3, examples of functional (services provided by the system) and non-functional requirements (constraints on these services) and the captured application/system features are shown.

Another part of the repository is the "Dictionary", which is indicated using blue color in Fig. 9. The requirements defined in "Reference" part should be related to the security requirements already saved in the repository's "Library" part. "Dictionary" contains the names of distinctive types of possible user profiles, different operating systems, distinctive authentication mechanisms and access control systems, which are known or expected to have security vulnerabilities. Filling in the dictionary part is the relatively easy part of building the repository and it does not take an excessive amount of time. It requires security and system knowledge. Yet, once it is complete, maintenance does not require an excess of exertion. This part would not change much from project to project. In Fig. 9, it can be seen that application/system features are partitioned into different groups. These components/features are all related to the main application/system in some way. An application/system may be related to one or more application system components and an application/system component may be related to one or more application. Once the "Reference", "Dictionary" and "Library" parts are filled in, the best possible application/system components, class types selected from "Dictionary" part of the repository will be used to capture actual security requirement templates from "Library" part.

"Library" part of the repository, shown using the green color in Fig. 9, is where security requirements templates are stored. The requirements are expressed using short sentences in natural language. The security requirement templates are called "Requirement Template" instead of "Requirement" only. The reason is the requirements are expected to have parts to be adopted/ changed/specialized before using in the "New Project". It is up to the user to mark or not mark such parts using special characters such as "<, >" or "||".

Every application/system feature may have its particular weaknesses, or it might be directly associated to a known kind of security vulnerability. So as to resolve these vulnerabilities or threats, a security requirement ought to be added to the repository. Filling in this part of the repository is a time consuming and continuous process. The repository is expected to grow with each new project. The requirements can be related to one or more application/system features in the library part of the repository. These relations should also be stored in the system. Once the vulnerability or the actual threat is defined, other template-based models such as the CC can be used to get an advantage of its already written templates in a formal manner. Books, vendor sites, best practice guides, standards, and precaution sets from risks analysis studies

TABLE 3. Functional and non-functional requirements and related application/system features.

Requirement	Instance Name of Application/ System Feature	Application/ System Component Class Type
The information software and technology systems on which the transfer or export takes place shall be such that all transfers or exports are protected from being received by any unauthorized users.	• Any Authentication Mechanism • Data Transfer • Data Export	• C-Authentication Mechanism • H-Functionality • H-Functionality
The system shall secure the data with electronic signatures.	• Uses Electronic Signature	• T-Tag
The system shall allow user to upload files.	• File Uploading	• H-Functionality
The dashboard frontend shall be implemented using: .Net/C#/ASP.NET/ASPX page using HTML5 and JavaScript technologies. Technologies like Silverlight or Flex / Flash shall not be considered.	• .Net/C#/ASP.NET/ASPX • HTML5 • Java script	• J-Language • J-Language • J-Language
The dashboard shall be displayable by calling it through a URL (including parameters).	• Web Based Application • Any Browser • Any Network Usage	• B-Application Architecture • E-Browser • L-Network

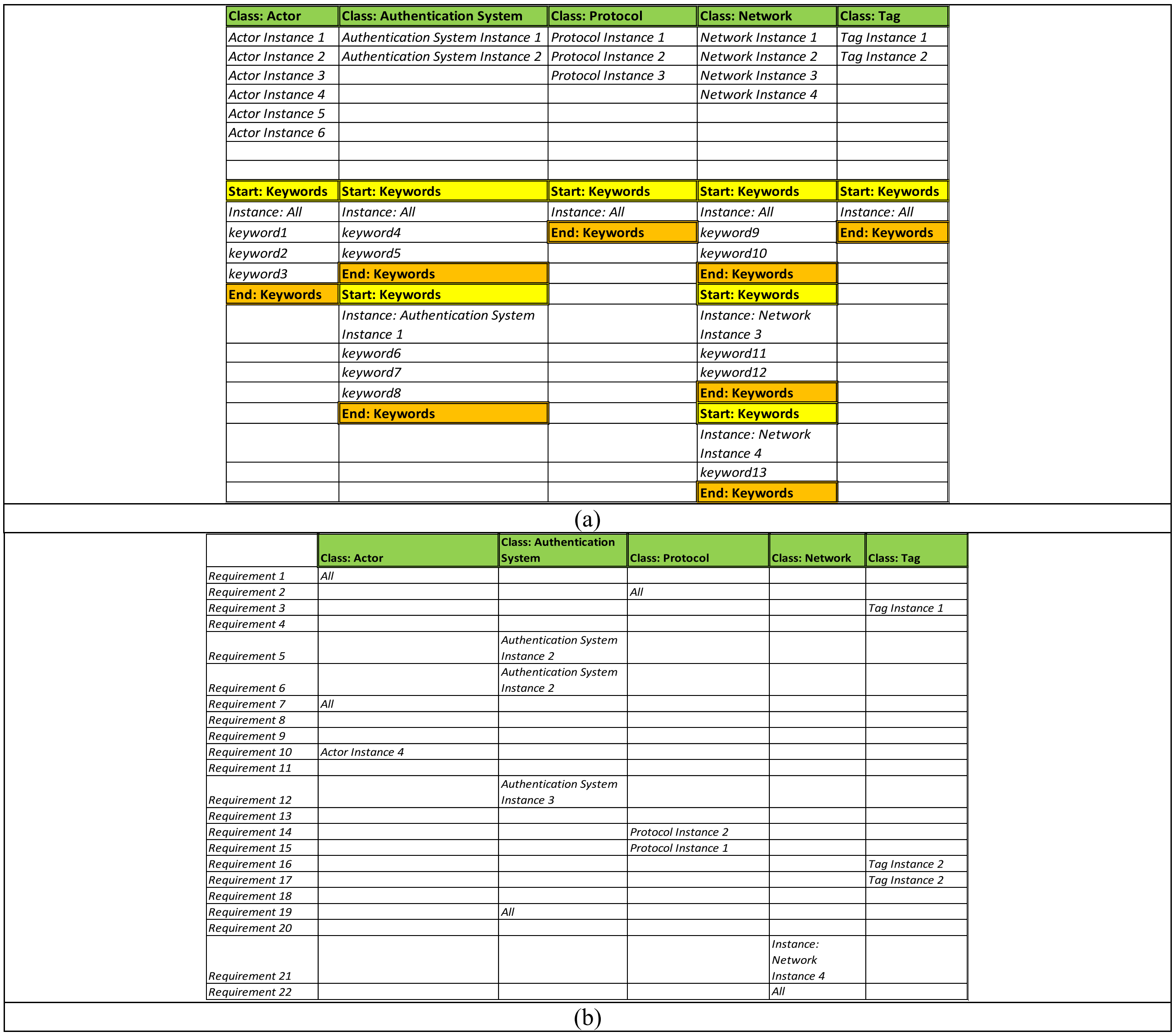
can be the source for this part. As the number of resources increase, library will grow. Requirement statements can be filtered based on creation date, source/origin, and creator. These fields can also be used to delete old requirement statements which belong to old dates or expired sources. Once the "Library" part of the repository is completed, security requirements, which will resolve the issues for a new project can be gathered through relating the non-security requirements to the security requirements over application/system features. The most effortless approach to do this is searching the application/system features in the SRS document. This search can be done automatically or manually. Natural language processing tools have been used before on SRS documents to increase reusability and automate system design [40].

Sometimes, the exact name of the "Dictionary" element in the SRS document cannot be found during a manual search or programmatically. For example, in the SRS file there may not be a term called "Bell La Padula Access Control Mechanism" or "Multilevel Access Control System", but instead a different wording might have been used such as "There should be a security access control mechanism which is based on clearance." Therefore, in order to catch the actual application/system components, searching only for application/system components from the "Reference" part is not enough. Having a set of keywords related to application/system features, security subjects, domains or technologies will both improve the search results and the quality of the security sets elicited from the Security Requirements Repository. For this reason, a new class called "Keyword" is added to the repository in the second version of this model. In the UML diagram, it can be seen that every application/system component may have one or more keywords. For the majority of the application/system components, these keywords may simply be anonymous, or similar terms or abbreviations for a technology that may help to discover the requirement related to that specific application/system component such that Windows-Microsoft Windows, IDS - Intrusion Detection System - Intrusion detection and Prevention System. For other technologies holding a set of keywords which commonly exist in the textual descriptions of that technology may help discoveries, such as Biometric Authentication -Retina-Iris, Cryptographic Function-Hash-Key Length, and Physical Protection-ISO 27001:2013 -Fence-Camera. Besides common knowledge, a more formal approach such as calculating the exact frequency of occurrence of each word in a technically accepted document covering the subject, and then, using manual inspection to make a choice of keyword sets related to the chosen subject, or using concept map diagrams such as Leximancer [41] for the latter group would be beneficial. In Fig. 4, generation and use of application/system components and keywords is included in the timeline of activities.

C. NORMALIZED REPOSITORY STRUCTURE AND PYTHON BASED REPOSITORY SEARCH ENGINE

As described in the methodology section, the data structure shown in Fig. 9 in the Appendix Section was transferred to a spreadsheet design. Top-level relationships of this diagram are provided in Fig. 3. In this design, the functional and non-functional non-security requirements, "Reference Part" is expected to be in a txt file. Fig. 5 (a) presents a sample storage of the "Dictionary Part". Fig. 5 (b) represents the information structure related to the "Library Part". In Fig. 5 (a) there are 5 classes, each having a variable number of instances. For some of the classes and instances, the sets of keywords are assigned. In Fig. 5(b), Requirement 1 is associated with all instances of the first class, Requirement 3 is associated with Instance 1 and Instance 3 of the second class, and Requirement 8 is associated to all instances of second class and only instance two of the third class.

Table 7 in the Appendix Section includes the repository search engine Python code. Shortly, the search engine creates a dictionary of class-instance associations. It also holds the instance-keyword associations in another structure. It checks the existence of all the words from the available Reference document holding the non-security requirements, and stores the result in a Boolean dictionary. Later, it reads all the reusable requirements from the Library Part into a dataframe.



Class: Actor	Class: Authentication System	Class: Protocol	Class: Network	Class: Tag
Actor Instance 1	Authentication System Instance 1	Protocol Instance 1	Network Instance 1	Tag Instance 1
Actor Instance 2	Authentication System Instance 2	Protocol Instance 2	Network Instance 2	Tag Instance 2
Actor Instance 3		Protocol Instance 3	Network Instance 3	
Actor Instance 4			Network Instance 4	
Actor Instance 5				
Actor Instance 6				
Start: Keywords	Start: Keywords	Start: Keywords	Start: Keywords	Start: Keywords
Instance: All	Instance: All	Instance: All	Instance: All	Instance: All
keyword1	keyword4	End: Keywords	keyword9	End: Keywords
keyword2	keyword5		keyword10	
keyword3	End: Keywords		End: Keywords	
End: Keywords	Start: Keywords		Start: Keywords	
	Instance: Authentication System Instance 1		Instance: Network Instance 3	
	keyword6		keyword11	
	keyword7		keyword12	
	keyword8		End: Keywords	
	End: Keywords		Start: Keywords	
			Instance: Network Instance 4	
			keyword13	
			End: Keywords	

(a)

	Class: Actor	Class: Authentication System	Class: Protocol	Class: Network	Class: Tag
Requirement 1	All				
Requirement 2			All		
Requirement 3					Tag Instance 1
Requirement 4					
Requirement 5		Authentication System Instance 2			
Requirement 6		Authentication System Instance 2			
Requirement 7	All				
Requirement 8					
Requirement 9					
Requirement 10	Actor Instance 4				
Requirement 11					
Requirement 12		Authentication System Instance 3			
Requirement 13					
Requirement 14			Protocol Instance 2		
Requirement 15			Protocol Instance 1		
Requirement 16					Tag Instance 2
Requirement 17					Tag Instance 2
Requirement 18					
Requirement 19		All			
Requirement 20					
Requirement 21				Instance: Network Instance 4	
Requirement 22				All	

(b)

FIGURE 5. Spreadsheet design which suits repository model and can be parsed via the search engine.

Using this data-frame and the existence dictionary, it forms a document holding a list in the form Class Name-Instance Name-Requirement Text. During requirement engineering practices, the user can use the list provided by the tool, elicit the requirements to build the final list of necessary security requirements.

V. EMPIRICAL STUDY DESIGN FOR VALIDATION

The empirical study designed for validation had two parts. In the first part, to examine the results of using a security repository, a sample repository library was prepared using various resources. Then, this repository was used to determine security requirements for two case studies selected from two published articles. The aim was not to prepare a complete repository with complete sets of keywords. However, even having a sample repository was sufficient to demonstrate the usage and the benefits of the model.

In the second part, a workshop was designed to enable the evaluation of the proposed model by participants from the industry.

Potential threats to the validity of the validation study are related to biased or incorrect selection of validation resources and participants and the participants' own bias. In order to eliminate these threats, the sample repository which will be described in the following section is formed using not a single source but multiple sources from different origins. Although the workshop which is part of the validation is a time taking process with multiple stages, the number of participants is kept moderately high having different level of experiences. Bias of the participants during the surveys are eliminated by keeping the proposed system and all the other compared systems completely anonymous. The details of the preparation, selection, conduction and evaluation for the validation study are described in the following sections.

A. PREPARING A SAMPLE SECURITY REPOSITORY

The sample repository consisted of the library part of the requirements with selected application/system features, and keyword sets and the security requirements related to the application/system features from the library part. On Github a project named "Security-Requirements-Repository-Through-Application-System-Components-and-Keywords" was created to store sample repository information, the case study inputs and outputs. The list of selected application/system components for the sample repository can be found on this Github project*. Keyword sets were generated by word frequency analysis. As specified in the Introduction Section, preparing a full security repository is beyond the scope of this article.

To prepare the repository, several sources were used. Initially, some SRS files [42]–[44] were considered. The requirement sets obtained in this way, however, were insufficient to be taken as a base. This observation also supports the initial opinion that writing a good SRS which has all the necessary security requirements is a very time consuming and difficult task. Next, the CC security documents [7], [8] were examined resulting with requirements that are very generic and requiring some extra work before usage in an SRS. However, the requirement templates from the CC were added to the sample repository and related to application/system components to the library part. Finally, some books [45]–[49] focusing on information security were used. Mostly, handbooks related to well-known security problems were selected, as they were already targeting most of the known security threats and providing the requirement statements to resolve them. As a result, 422 security requirement items were stored in this security repository.

B. SELECTION OF SAMPLE REQUIREMENT SETS FOR COMPARISON

Designing a new software requirement scenario as a case study subject to show the advantages of the proposed solution would possibly include a positive bias for the proposed model. Therefore, the authors queried the literature to find published sets of requirements which include both functional and non-functional requirements and has a length which is suitable for validation purposes. As a result of this query, two published articles were found, which are summarized in the next section. Although, it is not claimed that the given requirement sets are the results of detailed analyses, they were found useful for applying the proposed model to see the difference it makes. The time and effort spent for the security requirements analysis were not mentioned in these articles. Knowing that even in real life SRS documents, the security requirement analysis results may be limited, causing either incomplete requirement sets or very high-level abstraction leaving the details to the design phase of the project, it seemed to be not wrong to use the scenarios and corresponding requirements sets from selected articles for validation purposes.

The first case study was the "CrowdRequire". For this case, an analysis of the requirements of a Crowdsource application, named CrowdRequire [43] was made. This is a platform that supports requirements engineering using the crowdsourcing concept. In the paper named "CrowdRequire: A Requirements Engineering Crowdsourcing Platform", functional and nonfunctional requirements are provided. However, these requirements did not convey much detail, since their motivation was not preparing security requirements, which was expected. Github project lists the functional and non-functional requirements taken from the CrowRequire article [50].

The second case study was "Internet of Things Application" [51]. In the article named "A survey on Internet of Things", Alqassem and Svetinovic [52] made a detailed analysis of the Internet of Things subject. They have also published an article named "A Taxonomy of Security and Privacy Requirements for the Internet of Things" [52].

C. DESIGN OF THE VALIDATION WORKSHOP

The workshop was planned like a lecture related to frameworks and techniques for identifying security requirements. Evaluation workshop had three main parts. In the first part, the demographics information of the attendees was queried, including their experience on both requirements analysis methods and on information security concepts.

In the second part of the workshop, comparison of the security-focused requirement analyses methods was made. In this part, six security requirements methods were described to the attendees briefly, including their highlights, basic steps, and key points, sequentially through the use of power point slides. After the description of each method, five points Likert scale questions were asked to the attendees separately for each method.

In order to avoid causing a positive or negative bias for any of the methods, no information was given related to the actual names, origins, and creators of the methods including the proposed method. An order number and a short description were assigned for each method for identification. For example, the Common Criteria based method was just pronounced as "Method 2: Use of a Generic Requirements Framework", the proposed method was pronounced as "Method 4: Use of a Requirements Framework" so on. To enable a comparison of the methods and the techniques they use, detailed information on these frameworks and techniques were shared with the participants, during a one-hour presentation session. This information included the techniques for data collecting, searching, retrieving, classification, and elicitation; the description of the data used and or stored for each method; community and tool supports (if exists); information related to standards supporting each method (if exist); key points and highlights; graphical illustrations, such as graphs describing data or repository structures, or workflow showing phases of the techniques.

As a base method for comparison, the traditional requirements engineering lifecycle with no requirements repository or reuse was also included under the name of the traditional method. All of the participants were familiar with traditional requirements analysis methods and use-case based techniques. Only very few of them had little familiarity with the OWL data structures and tools. About, half of the attendees also had familiarity with the Common Criteria standard as an international security evaluation standard. The rest of the requirement reuse based methods including the proposed method were unknown for the participants.

In the third part of the workshop, application of the proposed method was carried out in the computer laboratory. The description of the first case study topic, a set of sample keywords, and a sample repository were given to the users and users were requested to apply the steps of the proposed method to the sample case in about 60 minutes. Prior to the application of the method, the steps of the proposed method were described to the participants again in two sessions. 29 participants (14 and 15 participants from session one and two) formed ten groups of 2 to 3 members and each group independently applied the proposed method and submitted their results. After this workshop, each attendee was asked to answer questions related to the applicability of the proposed method and their experience independently from other group members.

D. WORKSHOP PARTICIPANTS

The participants were all professionals taking evening education in a non-thesis graduate program that targets working individuals in the field. Respondents were working either in private sector or as civil servants in governmental departments with varying experience levels. The workshop has been conducted with 29 participants. The participants had varying levels of work experience, 51.7% junior, 31% senior, 10.3%, 6.9% owner or partner. The 34.4% of the participants were female, and 65.5% were male. All of them were working in IT companies besides their graduate studies. Fig. 6 shows the summary of their experiences in the IT sector, in information security area and in requirements engineering area.

VI. RESULTS

A. FIRST CASE STUDY RESULTS: USAGE OF SECURITY REPOSITORY ON CROWDREQUIRE WITH COMPARISON

For the first case study, the aim was to find the keywords matching the predefined application components and then the set of security requirements from the repository using these components. Table 4 shows the keyword analysis results, related to components, how the requirements are handled in the original document and the suggested security requirements, for the first case study.

B. SECOND CASE STUDY RESULTS: USAGE OF SECURITY REPOSITORY ON INTERNET OF THINGS APPLICATION CROWDREQUIRE WITH COMPARISON

For this case study, the keywords and new application/system components were identified and compared with the results of Alqassem and Svetinovic [52]. The aim was to extend their outcomes with the assistance of a security repository modeled in the proposed way. The original security requirements for this case (existed in Github project) were selected after a detailed vulnerability analysis and elicitation explained in the Alqassem and Svetinovic's [52] article. From the problem definition article, the keywords were selected. Using the keywords, the corresponding application/system components listed in Table 5 were obtained. Evaluation was made based on two factors. The first is whether the application/system component was new to the sample repository or it already existed. This shows how a sample repository grows. The second is whether the security requirements corresponding to the keywords and application components have already existed in the original security requirement set or not. This shows the degree of success of the proposed method.

C. RESULTS OF METHOD COMPARISONS IN THE WORKSHOP

For the comparison of the methods six questions were designed with five-point Likert scales. Fig. 7 shows the questions and mean values for all the answers given by all participants.

One sample one sided paired t-tests were made for all 29 records in a single group to measure the significance of the differences of the mean values between method 4 (proposed method) and the other methods at 90% and 95% significance levels. Looking at the statistical results, in terms of suitability of handling security requirements, the proposed method is similar to the other methods except for the traditional method. Regarding all other listed criteria, such as the capability of using previous requirements, or feasibility of completion in the time allotted, the proposed method had significant advantages over other methods at 95% or 90% levels with a few pair-to-pair comparison exceptions.

The key findings of validation questionnaire are as follows:

- The suitability of proposed method (Method 4) for small companies has been found quite high as proposed by the authors.
- The only method which has higher suitability for small companies than the proposed method (Method 4) was the traditional requirements elicitation method (Method 1) which is logical, because it requires no repository preparation, thus, less resource.
- Completion in the time allotted was found the most feasible for the proposed method (Method 4). This result is in line with the authors' initial claims and indicate that the participants have found that the proposed method (Method 4) as practical and easy to use.
- The OWL based method (Method 5) has been found to be the least feasible in terms of time consumption. The reason for this may be the fact that the participants were not familiar with OWL-based tools and techniques.
- The second least feasible method was the Common Criteria based repository (Method 2), due to its complicated structure.

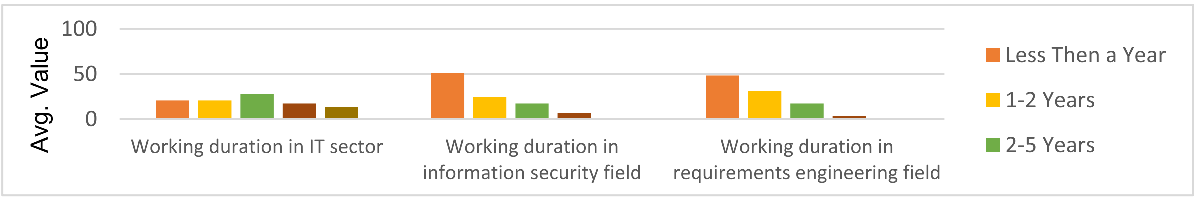


FIGURE 6. Workshop participants' experiences on relevant fields.

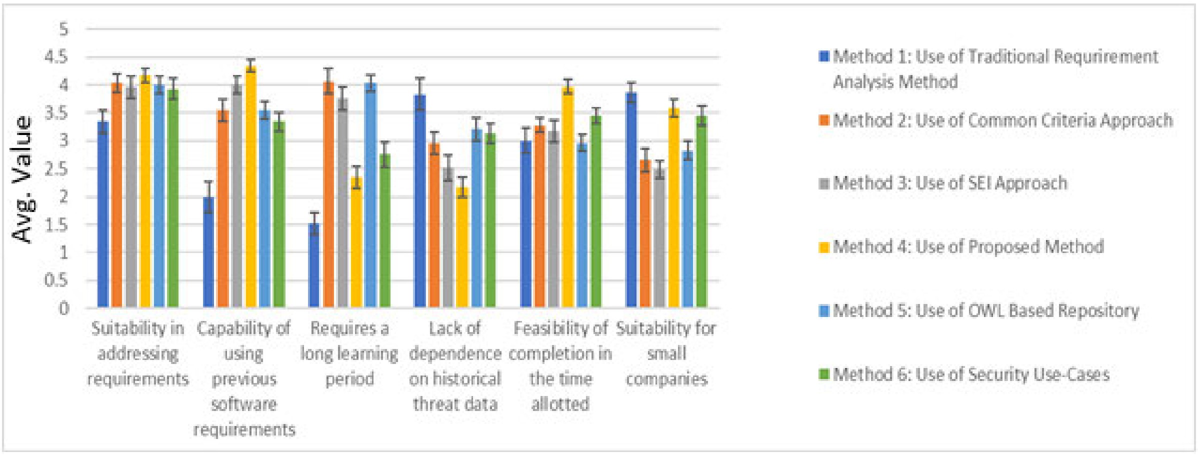


FIGURE 7. Results of method comparisons.

- In terms of lack of dependence on historical threat data, the proposed method (Method 4) has been found to be the lowest. The proposed method does neither have an explicit focus on threat data or threat analysis, nor a claim of being independent of historical threat data. However, the existence of the repository and selected application/system components/features eventually resulted in the idea of dependency of the previous threat data on the participants.
- In terms of suitability of addressing the requirements, the proposed method (Method 4) has the highest score followed by the Common Criteria based model (Method 2) and the method proposed by SEI (Method 3).
- The capability of using previous software requirements has also been found the highest for the proposed method (Method 4). Although some of the other methods also included the reuse of previous requirements, this result indicates that the participants have found the data structure of the proposed method easy and most suitable to store and retrieve the requirements.
- The proposed method (Method 4) has the second lowest score in terms of having a long learning period after traditional requirement analysis.

D. RESULTS OF PROPOSED MODEL APPLICATION EXPERIENCE IN THE WORKSHOP

The results of applying the proposed model to the sample case are shown in Fig. 8. The results are quite promising and show that participants have understood, benefited from, and applied the proposed model effectively for the sample case. The submitted keywords, component lists, and requirement sets for each group in 10 separate files were also examined and scored by the authors in order to determine the effectiveness of the proposed model and the limitations of the sample security requirements set. The summary of numerical results of using sample repository is shown Table 6.

Key points related to application of proposed method are as follows:

• The difficulties of the application of the method in the first session have been observed by the authors and this resulted in a better explanation of some issues during the second session.

- ○ The explanation of the steps of the proposed model has been improved after the first session.
- ○ Limitations of the sample repository such as not having enough requirement sets for each component type in the Excel file, has also been better explained to the attendees of the second session.

• A better understanding of the method and limitations of the workshop resulted in a higher average score for the majority of the comparison metrics for the groups of the second session.

• The attendees of the second session gave more positive answers to the third part of the survey compared to the attendees of the first group.

TABLE 4. Keyword analysis results for "Crowdrequire" [50].

Keywords: register, registration**, Instance (Class):** Authentication Mechanism (C-Authentication Mechanism)
- "The application must log the user out when the user is exits" [42]
- "The system shall require each user to be successfully authenticated before allowing further actions on behalf of that user" [7] [8].
- "Logout must be automatic after some fixed period of time of inactivity." [42]
- "The system should detect and log unsuccessful authentication attempts" [7]-[8].

Keywords: payment, charge, cash, reward**, Instance (Class):** Payment System (N-Payment System)
- "Cryptographic hash algorithm should be used when generating session keys in a network application. Using incrementing numbers or timestamp may cause an attacker guessing the key" [46].

Keywords: web based, web 2.0**, Instance (Class):** Web Based Architecture (B-Application Architecture)
- "Database should be tested (by passing client application) via direct web proxy for the possibility of SQL Injection Attack Pattern in web-based applications" [46].
- "Application should be verified against cross site scripting which results an attacker's script to execute in a web browser" [46].
- "Client-side code should be verified separately for the situation that unexpected inputs are handled correctly for the possibility of cross script attact or sql injection attack" [46].
- "Web server should be verified against poor buffer handling" [46].

Keywords: internet, **Instance (Class):** Internet Network (L-Networks)
- "Application design should not rely on single security mechanism in distributed or multitier applications. Different security mechanisms should exist in different levels to increase the reliability of the overall system" [46].
- "Consent is obtained before personal information is transferred to or from an individual's computer or other similar device." [49]
- "The system shall provide secure integrity-checking capabilities through the interface between the user and the system and among systems. It is necessary to ensure the validity of transmission between systems (i.e., to ensure that the data received is the same as that which was sent)." [49]

Keywords: platform owner**, Instance (Class):** platform owner, Actor (A-Actor)
- "The system shall be able to generate an audit record for the all the auditable events for the minimal level of audit" [7] [8].
- "The system shall take upon detection of a potential security violation" [7] [8].

Keywords: administrator, **Instance (Class):** administrator, Actor (A-Actor)
- "The application shall not divulge in clear text the static authenticator (e.g., password, PIN number, token seed, smart card seed, etc.) of one user to any other user, including administrators. Passwords shall not be transmitted, stored, or echoed in clear text. Administrators must have the ability to make changes to authentication information, but must not be able to easily impersonate the user." [49]

Keywords: customer, user**, Instance (Class):** User (A-Actor)
- "The privacy rule requires that organizations that engage in financial activity provide customers copies of their privacy policy and explain their practices on sharing customer information." [49]
- "The use of personal information in process and system test and development is prohibited unless such information is anonymized or otherwise protected in accordance with the entity's privacy policies and procedures." [49]

Keywords: access, authorization**, Instance (Class):** Access Control Mechanism (D-Access Control Mechanism)
- "The application supports multiple users with different levels of privilege. The application assigns users to multiple privilege levels and defines the actions each privilege level is authorized to perform. The various privilege levels need to be defined and tested. Mitigations for authorization bypass attacks need to be defined" [46].

Keywords: employee**, Instance (Class):** Employee Data (F-Data)
- "Personal information is disclosed to third parties only for the purposes described in the notice, and for which the individual has provided explicit consent, unless a law or regulation specifically requires or allows otherwise." [49]

Keywords: customer, **Instance (Class):** Customer Data (F-Data)
- "Notice is provided about the entity's privacy policies and procedures at or before the time personal information is collected." [49]

Keywords project, **Instance (Class):** Project Data (F-Data)
- "Individuals are informed about the choices available to them with respect to the collection, use, and disclosure of project information that implicit explicit consent is required to collect, use, and disclose project information, unless a law or regulation specifically requires or allows otherwise."[49]
- "Project data no longer retained is anonymized, disposed of, or destroyed in a manner that prevents loss, theft, misuse, or unauthorized access."[49]

Keywords: messaging platform, communication tool**, Instance (Class):** Messaging Functionality (H-Functionality)
- "The system shall provide mechanisms to detect communication security violations in real-time, such as replay attacks that duplicate an authentic message. Serious attacks on data integrity, such as replay, represent a significant threat to system security and need to be dealt with as soon as possible."[49]

Keywords: submit task**, Instance (Class):** File Upload, Data Insert (H-Functionality)
- "The application accepts input from the user and saves the input in the database using SQL language. SQL injection mitigation is required" [46].
- "The application manages sessions for a logged-in user, and session hijacking mitigations must be in place." [46].
- "User input must be validated for length and characters." [46].

Keywords: crowd**, Instance (Class):** Crowd Application(T-Tag)
- "There will be at least two active security officers among the users" [45].
- "These security officers should have CISSP certificate. They should be volunteers and they will be selected by the administrators and owners of the system" [45] .

Keywords: browser, **Instance (Class):** Browser (E-Browser)
- "The system shall work on the latest versions of IE, Firefox, Google Chrome and Mozilla browsers" [45].

Keywords: desktop, Instance (Class): Desktop PC (I-Hardware)
- No corresponding requirement exist in the sample repository

Keywords: mobile, **Instance (Class):** Mobile Device (I-Hardware)
- No corresponding requirement exist in the sample repository
- Application Specific API (Application Programming Interface) (B-Application Architecture)

* This table shows how the keywords are related to class instances and the selected classes and how in the end they helped to find the listed requirement from the repository

TABLE 5. Keyword analysis results for the problem definition of Internet of Things [51].

Keywords:RFID, Sensor, Actuator, Radio Frequency Identification, Tag, Microchip, Antenna, Electronic product code, EPC, Global, **Component:**RFID (L-Network), **Evaluation:**New in repository. Requirement is not handled properly in the original set. No specific security requirement is mentioned related to the vulnerabilities of RFID.
Keywords:NFC, Near Field Communication, **Component:**NCF(L-Network), **Evaluation:**New in repository. Requirement is not handled properly in the original set. No specific security requirement is mentioned related to the vulnerabilities of NFC.
Keywords:M2M, Machine to Machine Communication, **Component:**M2M (L-Network), **Evaluation:**New in repository. Requirement is not handled properly in the original set. No specific security requirement is mentioned related to the vulnerabilities of M2M.
Keywords: V2V, Vehicular to Vehicular Communication, Intelligent Transport System, ITS, **Component:** V2V(L-Network), **Evaluation:**New in repository. Requirement is not handled properly in the original set. No specific security requirement is mentioned related to the vulnerabilities of V2V.
Keywords: Wireless, **Component:**Wireless (L-Network), **Evaluation:**Exists in the repository. Data integrity is mentioned in the text but no special precaution exists for wireless users.
Keywords: Mobile Phone, **Component:**Mobile Phone (S-User Interface Device), **Evaluation:**New in repository. Data integrity is mentioned but no special precaution exists for mobile phone users.
Keywords: Manufacturing Industry, Service Industry, **Component:**Manufacturing Data, Service Data (F-Data) (Q-Regulation), **Evaluation:**Partially exists in the repository. No regulation rules are mentioned for the listed industries and no specific type of security precaution is mentioned for the protection of valuable production data.
Keywords: Hacker, **Component:**Hacker (A-Actor), **Evaluation:**Exists in the repository. Requirement is handled in the original set. However, if the type of expected hackers were given in the security requirements, these would improve the precautions.
Keywords:communication, transaction, tracking, monitoring, control, prevention, inventory management, mobile payment, **Component:**Communication, Transaction, Tracking, Monitoring, Control, Prevention, Inventory management, Mobile payment (H-Functionality), **Evaluation:**Partially exists in the repository. Requirement is partially handled in the text. Specific attention should be given to mobile payment regarding security.
Keywords: Client, **Component:**Client (A-Actor), **Evaluation:**Exists in the repository. Not handled in the text.
Keywords: IPV4, IPV6, TCP, **Component:** IPV4, IPV6, TCP (P-Protocol), **Evaluation:**Exists in the repository. Not handled in the text.
Keywords: Proxy Attack, Man in the Middle Attack, Snooping, **Component:**Proxy Attack, Man in the Middle Attack, Snooping (T-Tag), **Evaluation:**New in Repository. These types of attack possibilities for the application were mentioned explicitly in the text.
Keywords: Personal Information, **Component:**Personal Data (F-Data), **Evaluation:**Exists in the repository. Requirement is handled in the text.
Keywords: Geographical Source Routing, GPRS, **Component:**Geographical Source Routing GPRS (P-Protocol), **Evaluation:**New in repository. Not handled. Specific attention is given to GPRS regarding security.
Keywords: On Demand Routing Protocol, **Component:**On Demand Routing Protocol (P-Protocol), **Evaluation:**New in Repository. Not handled. Specific attention should be given to specific protocols regarding security.

* This table shows how the keywords are related to class instances and the selected classes and how in the end they affected to find the requirements and to update the repository

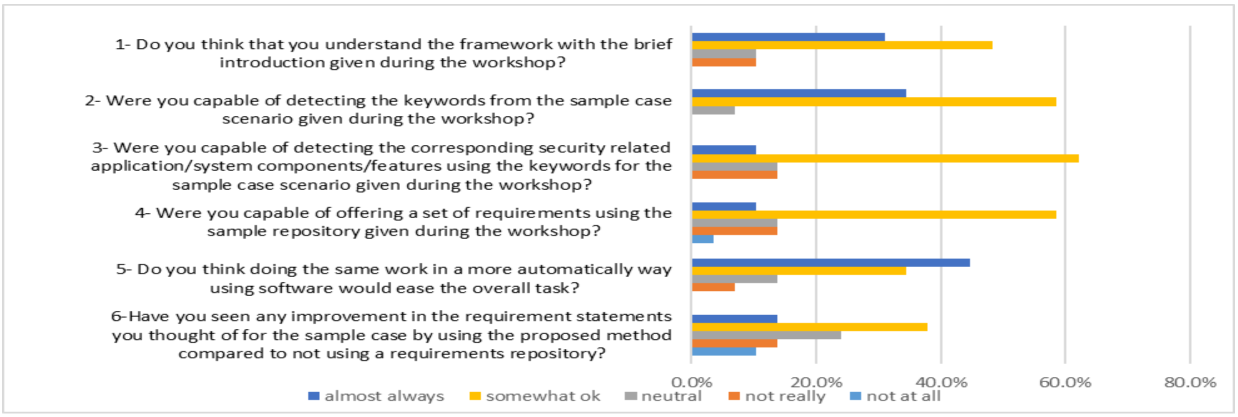


FIGURE 8. Workshop results of application of proposed method.

- Since the provided sample repository was not complete, all groups were consistently able to offer the necessary requirement statements related to some parts, such as the web server, but were not able to offer much for some other parts which were missing in the sample repository. This result may indicate that all the groups understood and used the technique

TABLE 6. Workshop numerical results.

	5 Files from Session 1	5 Files from Session 2
Average number of total requirements	25.2	25.8
Used relevant keywords	2/5	5/5
Keywords are relevant	Partially	Yes
Average number of different new requirements	1	0.41
Average number of missing requirements	1.66	2.29
Improved repository by suggesting new requirements	1/5	0/5

in a similar way and achieved similar results independent of each other.

• Very few of the attendees were able to offer new requirement statements for the sample case due to the limited time, which is reasonable because initial forming of the requirement statements requires a time taking process.

VII. DISCUSSION

In this part, design decisions, limitations, and the detailed results of using the proposed method are depicted. The proposed model stores text-based requirement statements in natural language. The number of requirements presented or expressed in those ways in such earlier studies does not exceed ten at most, missing many important security issues. With the proposed repository, using the mentioned resources was relatively easier due to the already mentioned benefits of text-based requirements. Hence, even the sample repository created for the demonstration purpose included 422 requirement statements, handling numerous security issues.

The repository has a structure which guides the user about what kind of information to be reused. This structure enforces the reuse of various kind of requirements. Thus, the reused information is not limited to a small common set such as, requirements related to the user authentication or cryptography implementation.

Flexibility is an advantage of the proposed model. The model allows definition of requirements in any precision level. Requirements stored in the repository can be as generic as the Common Criteria case, but it is also possible to store specific requirements coming from other acceptable sources, which has been the case for the sample repository. The consistency of the requirement statements is important for high reusability. Since multiple sources are planned to fill in the repository, it may be difficult to achieve the same level of consistency for all the requirement statements. For example, the requirement statements taken from the Common Criteria documents include ‘‘assignment’’ parts; other sources include parenthesis which indicates parts that should be filled in by the users. Repository creators may decide on a representation for these situations. The use of repository also allows grouping the requirements according to their functionality, such as in authentication and access control. This grouping mechanism would also help to resolve inconsistencies which would otherwise cause vulnerabilities. Including additional information, besides the actual security requirement statements, is also beneficial to increase the usability of the repository. In the sample repository, the rationale of the requirement is included in parenthesis for this purpose.

Creation of a sample repository and application of the model for two cases pointed out two important concerns. Multiple reputable sources of security requirements should be used when forming the repository, because each source has its own strengths and there is a necessity to be careful when using and selecting the keywords. Although some ways have been offered to easily ameliorate the generation of keyword sets, the selection of keywords for specific application/system components may be subjective. Repository creator team should be taught to eliminate meaningless keywords and improve the remaining ones.

In the results section, demonstration was made for both case studies, showing that the existence of a security repository in the proposed model can make significant improvements on the original security requirement sets. In the first case, although the original requirement set lacked details, it pointed out the necessity of top level confidentially, access control, integrity, prevention of data loss and availability requirements. Original requirement analysis lacked a detailed security analysis, which is usually the case in real life due to limited resources or time. Usually, functional requirements are worked on more properly than security requirements during the SRS phase. Security requirements are only declared using some fundamental set of sentences and are expected to be improved in later phases of the project. As a result of the application of the proposed method, some security requirements from the sample repository were offered, resulting in a detailed set of security requirements covering some, probably not all, important security issues. In the second case, a manual keyword analysis was made which resulted in a very large set of possible application/system components. Next, original requirement sets have been checked to see if they meet all the security needs. This analysis showed that even detailed security analysis results may lack important security requirements and the usage of the repository handles these situations effectively.

Even using a few resources found online allowed us to create and improve the repository by adding some very important application/system components and corresponding security requirements during the process. For an organization which works on different IT problems continuously, and repeatedly, creating such a repository with certain quality would be relatively straightforward.

In terms of usability of the proposed system, during the workshop, the attendees managed to find out keywords and corresponding application system features. Later, Excel program filtering feature was used to find the requirements corresponding the application/system components. Even this limited functionality allowed the completion of the workshop for all groups. Having the tool support, repository search engine, boosted the usability of the overall system.

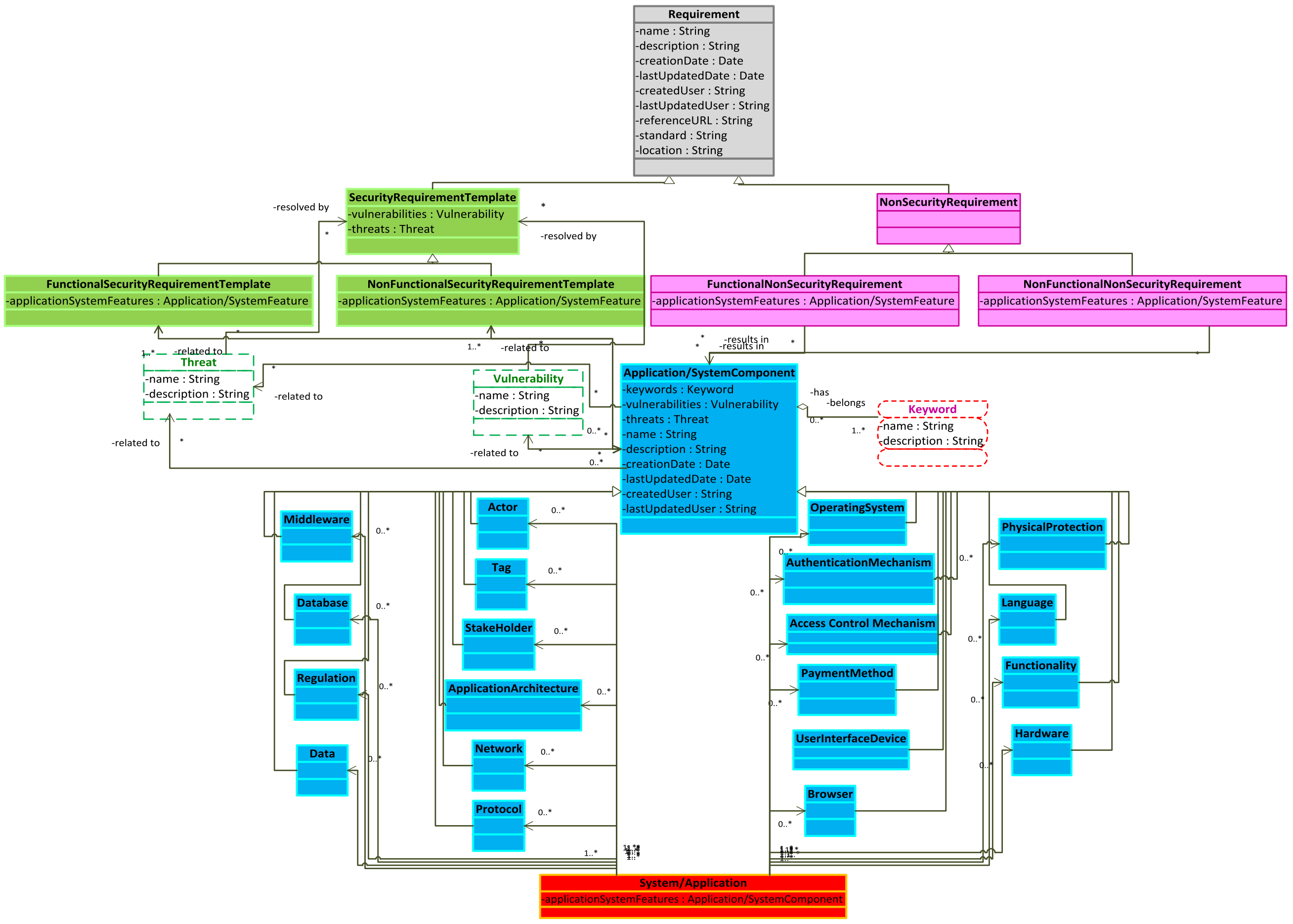


FIGURE 9. **Class diagram for the proposed model.**

Going into the details of using the sample repository, in order to compose associations with the components, it may be necessary to defragment the original requirement phrases to a level which better associate with the application/system components. For example, a requirement which points out the privacy of multiple data types may be fragmented to associate with each data source independently. The relation of requirement statements to some components may not be absolute for some requirement statements. For such situations, to be on the safe side, during the preparation of the sample repository, such requirements were associated with the appropriate components and the decision of including or excluding was left to the requirements elicitation phase. For example, the requirement of enabling backup of the security information was associated with all types of access control systems. However, ACL systems which do not store such information may not have to implement this requirement. This decision is left to the users of the repository.

An important finding achieved during the preparation of the sample repository was that the Tag class was also useful to point out security targets, such as confidentially and integrity, although initially it was designed to be used for relations which were not application/system components/features. Some requirement statements may have high associations to security targets. For example, the association to integrity is clear for some requirements pointing out the necessary data integrity checks during data entry. While preserving the associations with security targets using the Tag class, new associations to user interface devices, data types or data entry related functionalities can be built.

Another finding was also related to the Tag class. This Tag class was used to indicate use-cases whenever applicable. As mentioned earlier about the requirements which are related to security targets, some requirements have high relations to use-cases. For such requirements building associations to use-cases over the Tag class may also be beneficial. In the sample repository, use cases, such as session timeout, system login, and secure recovery were pointed out explicitly. Use of Tags to point out use-cases causes some resemblances to security patterns which explicitly point out the situations where the security solutions should be applied. One other resemblance to security patterns is due to grouping the requirement sets using application/system components. Security patterns have been categorized using

TABLE 7. Code for repository search engine.

```
import pandas as pd
from pandas import DataFrame


def get_start_keywords_row_index(df_col: DataFrame):
    """ get the first start_keyword position counted by index in df_col """
    first_start_keyword_idx = df_col[df_col == "Start: Keywords"].index[0]
    return first_start_keyword_idx

def extract_column(start_key_pos: int, df_col: DataFrame, classname: str):
    """ output list of instances names, keywords, mapping of instances -
keywords """
    set_instances = set()
    set_keywords = set()
    ingredient_instance_keyword = {}
    for col_num, item in df_col.iloc[: start_key_pos].iteritems():
        if item == "":
            break
        item = item.lower()
        set_instances.add(item)
    recording_key = False
    instance_prefix = "instance: "
    current_instance = None
    for col_num, item in df_col.iloc[start_key_pos:].iteritems():
        if item == "":
            continue
        if item == "Start: Keywords":
            recording_key = True
            continue
        if item == "End: Keywords":
            current_instance = None
            recording_key = False
            continue
        item = item.lower()
        if recording_key:
            if instance_prefix in item:
                instance_name = item.split(instance_prefix)[-1]
                set_instances.add(instance_name)
                current_instance = classname + ":" + instance_name
                ingredient_instance_keyword[current_instance] = []
                continue
            if current_instance is not None:
                set_keywords.add(item)
                ingredient_instance_keyword[current_instance].append(item)
            else:
                print(f"SEVERE PROBLEM: KEYWORD {item} is not
belonging to any Instance. Do re-check Sheet file please.")
    return set_instances, set_keywords, ingredient_instance_keyword

def read_cls_inst(filename: str):
    df = pd.read_excel(filename, headers=None, sheet_name="Classes-
Instances-Keywords")
    df = df.fillna(value="")
    orginal_classes_names = [classname for classname in df.columns]
    classes_names = [classname.lower() for classname in df.columns]

    mapping_classname_instance = {}

    all_keys = set(classes_names)
    all_ingredient_instance = {}

    start_key_pos =
get_start_keywords_row_index(df[orginal_classes_names[0]])
    for classname in orginal_classes_names:
        set_instances, set_keywords, ingredient_instance_keyword =
extract_column(start_key_pos, df[classname], classname.lower())
        mapping_classname_instance[classname.lower()] = set_instances
        all_keys = all_keys.union(set_instances)
        all_keys = all_keys.union(set_keywords)
        all_ingredient_instance.update(ingredient_instance_keyword)
    return all_keys, all_ingredient_instance, orginal_classes_names,
mapping_classname_instance

def read_text_content(filename):
    with open(filename, "r", encoding="utf-8") as f:
        content = f.read()
    return content

def is_instance_in(classname: str, instance_name: str, all_ingredient_instance:
dict, is_in_text_file: dict):
    """ check if instance is in the text file by checking its name or its ingredients
"""
    if instance_name not in is_in_text_file.keys():
        return False
    if is_in_text_file[instance_name]:
        return True
    combined_key = classname + ":" + instance_name
    if combined_key not in all_ingredient_instance.keys():
        return False
    for keyword in all_ingredient_instance[combined_key]:
        if not is_in_text_file[keyword]:
            return False
    return True

def check_all(classname: str, all_ingredient_instance: dict, is_in_text_file: dict,
mapping_classname_instance: dict):
    if is_in_text_file[classname]:
        return True, 'all'
    for instance in mapping_classname_instance[classname]:
        if is_instance_in(classname, instance, all_ingredient_instance,
is_in_text_file):
            return True, instance.lower()
    return False, ''

def read_requirement_file(filepath: str):
    df = pd.read_excel(filepath, sheet_name="Security Requirements")
    df = df.fillna(value="")
    return df

def append_row(output_file, row):
    with open(output_file, "a", encoding="utf-8") as f:
        f.write(row + "\n")

def main():
    keywords_file =
'D:\Dropbox\Security_Requirements_Repository_Makale\Sample
Repository\Sample_Repository_V10_Simplified.xlsx'
    output_csv = 'Output.csv'
    text_file =
'D:\\Dropbox\\Security_Requirements_Repository_Makale\\Sample
Repository\\2012-Svetinovic-AAAI.txt'
    print("I opened txt file")
    all_keys, all_ingredient_instance, orginal_classes_names,
mapping_classname_instance = read_cls_inst(keywords_file)
    is_in_text_file = {}
    text_content = read_text_content(text_file).lower()
    for key in all_keys:
        is_in_text_file[key] = f" {key} " in text_content or f"/n{key}" or
f"/n{key}" or f"/n{key}/n" or f" {key}" in text_content or f"{key} " in
text_content
    is_in_text_file["all"] = False
    df_requirements = read_requirement_file(keywords_file)

    with open(output_csv, "w", encoding="utf-8") as f:
        f.write("Classname,Instancename,RequirementText\n")

    for idx, row in df_requirements.iterrows():
        for classname in orginal_classes_names:
            cell_val = row[classname].lower()
            if cell_val == "":
                continue
            if cell_val == "all":
                exist, matching_instance = check_all(classname.lower(),
all_ingredient_instance, is_in_text_file, mapping_classname_instance)
                if exist:
                    append_row(output_csv,
f"{classname},{matching_instance},{row['Security Requirements']}")
            else:
                required_instances = cell_val.split(",")
                for required_instance in required_instances:
                    if is_instance_in(classname.lower(), required_instance.strip(),
all_ingredient_instance, is_in_text_file):
                        append_row(output_csv,
f"{classname},{required_instance},{row['Security Requirements']}")

    print(text_content)
if __name__ == "__main__":
    main()
```

many criteria to form groups of patterns so far in numerous studies.

Using Application Architecture class to store sub architec-tural decisions, such as the use of cookies or use of data entry forms, besides the high-level architectural types, such as the desktop application or web-based application, allowed classification of some particular requirements better, increasing the possible reusability during the preparation of the sample repository.

The proposed system has a few limitations some of which are inherent limitations due to the nature of the process, and others due to the design decisions. The proposed model does not include risk factors, risk analysis, and risk management. Risk analysis can be made using various methods at different levels in different organizations. This activity has not been included in the model on purpose to achieve simplicity and to allow the use of the model together with existing risk analysis techniques in an organization.

The proposed repository-based security requirement solution, similar to the mentioned studies in the literature review section does not guarantee completeness. The completeness of the security requirements should be checked together with the completeness of non-security requirements using existing requirements engineering practices. Customer evaluation and group decision-making techniques should be used to evaluate the completeness of both the repository items and the resulting security requirements. The number of the available attributes characterizing the security requirements is also a criteria of completeness for such kind of repository models.

Besides the completeness, the quality of the requirement statements from the sample repository can be questioned. The requirement sentences taken from the referenced sources were used as is in the paper to achieve a high level of ethical engagement. However, at least some of the sentences from the sample repository should be worked on to reduce the redundancy, increase uniformity and provide a consistent abstraction level for a real-world case. Some requirements from the sample repository should also be improved by adding a ''such that'' clause and should be continued with the exact details of the requirement in a real-world case.

Existing tool support can be carried to the web as a future task, to enable a crowdsourcing repository approach project. As the number of users of this repository increases, more sources can be included and the speed of growth may accelerate. One technological way of doing this is creating a SharePoint portal application. In the case of using a web interface with multiple users, editorial functionally, such as approving/rejecting the suggested keywords and requirement sets can be included for better functionality in the future.

VIII. CONCLUSION

In general, the efforts to characterize security requirements for a new software/system start from scanning possible security risks of the software/system. In order to do that, requirements engineers or system architects determine the threats to the system. In any case, it is challenging to find all the risks, at the broader view, of the software/system. This is one of the main reasons why security is not handled properly in most of the projects. The proposed model divides the system/software in a unique way to provide easy association with security requirement sources. Moreover, it is demonstrated and tested using two different case studies from published academic work and a sample repository. The results show that in the absence of proposed requirements repository, critical requirement statements would be missing from the listed requirements.

As future work, a public security repository can be created by cooperating with IT companies, which already have extensive SRS libraries. In the long term, it may provide a platform if a shared repository exists among developers doing similar projects. This cooperation with IT companies can likewise bring about better SRS libraries for them for internal usage. Some other useful attributes can also be added to the repository design, such as vulnerability analysis levels or function point measurements for different security solutions. Making more analysis on keyword sets and security concept diagram methods may result in finding new relationships among security concepts and this may also improve the data structure model of the repository.

Generation of keywords through the use of concept seed from any valuable security related source can be automated in the future. The generated concepts can be used to check existence of necessary concepts in an SRS, even without continuing with the definition of application/system components. However, mapping the resulting concepts to application/system components automatically requires some extra work which has not been done in this study. This may also be achieved by using a rule-based engine system for some extent.

APPENDIX

See Figure 9 and Table 7.

REFERENCES

[1] *ISO/IEC/IEEE International Standard—Systems and Software Engineering—Life Cycle Processes—Requirements Engineering*, IEEE Comput. Soc., Washington, DC, USA, 2018.

[2] F. Al-Hawari, M. Al-Zu'bi, H. Barham, and W. Sararhah, "The GJU website development process and best practices," *J. Cases Inf. Technol.*, vol. 23, no. 1, pp. 21–48, Jan. 2021.

[3] D. Mellado, C. Blanco, L. E. Sánchez, and E. Fernández-Medina, "A systematic review of security requirements engineering," *Comput. Standards Interfaces*, vol. 32, no. 4, pp. 153–165, 2010.

[4] A. Souag, R. Mazo, C. Salinesi, and I. Comyn-Wattiau, "Reusable knowledge in security requirements engineering: A systematic mapping study," *Requirements Eng.*, vol. 21, no. 2, pp. 251–283, Jun. 2016.

[5] *Information Technology—Security Techniques—Evaluation Criteria for IT Security—Part 1: Introduction and General Model*, ISO/IEC, Geneva, Switzerland, 2014.

[6] (Jun. 2016). *Common Criteria*. [Online]. Available: https://www.commoncriteriaportal.org/

[7] *Information Technology—Security Techniques—Evaluation Criteria for IT Security—Part 2: Security Functional Requirements*, ISO/IEC, Geneva, Switzerland, 2014.

[8] *Information Technology—Security Techniques—Evaluation Criteria for IT Security—Part 3: Security Assurance Requirements*, ISO/IEC, Geneva, Switzerland, 2014.

[9] D. Mellado, E. Fernandez-Medina, and M. Piattini, "A common criteria based security requirements engineering process for the development of secure information systems," *Comput. Standards Interfaces*, vol. 29, no. 2, pp. 244–253, 2007.

[10] M. Saeki and H. Kaiya, "Security requirements elicitation using method weaving and common criteria," in *Proc. Int. Conf. Model Driven Eng. Lang. Syst.*, Berlin, Germany, 2008, pp. 185–196.
[11] M. Amutio, J. Candau, and J. A. Mañas, "MAGERIT—Versión 3.0. Metodología de análisis y gestión de riesgos de los sistemas de información. Libro II-catálogo de elementos," Edita, Madrid, España, 2012.
[12] A. Toval, J. Nicolás, B. Moros, and F. García, "Requirements reuse for improving information systems security: A practitioner's approach," *Requirements Eng.*, vol. 6, no. 4, pp. 205–219, Jan. 2002.
[13] J. Lasheras, R. Valencia-García, J. T. Fernández-Breis, and A. Toval, "Modelling reusable security requirements based on an ontology framework," *J. Res. Pract. Inf. Technol.*, vol. 41, no. 2, p. 119, 2009.
[14] B. C. Zapata, J. L. Fernández-Alemán, A. Toval, and A. Idri, "Reusable software usability specifications for mHealth applications," *J. Med. Syst.*, vol. 42, no. 3, pp. 1–9, Mar. 2018.
[15] D. Firesmith, "Specifying reusable security requirements," *J. Object Technol.*, vol. 3, no. 1, pp. 61–65, 2004.
[16] C. Raspotnig, P. Karpati, and A. L. Opdahl, "Combined assessment of software safety and security requirements: An industrial evaluation of the CHASSIS method," in *Research Anthology on Artificial Intelligence Applications in Security*. Hershey, PA, USA: IGI Global, 2021, pp. 666–693.
[17] A. Zuccato, N. Daniels, and C. Jampathom, "Service security requirement profiles for telecom: How software engineers may tackle security," in *Proc. 6th Int. Conf. Availability, Rel. Secur.*, Vienna, Austria, Aug. 2011, pp. 521–526.
[18] I. Omoronyia, G. Sindre, T. Stalhane, S. Biffl, T. Moser, and W. Sunindyo, "A domain ontology building process for guiding requirements elicitation," in *Requirements Engineering*. Berlin, Germany: Springer, 2010, pp. 188–202.
[19] C. Salinesi, E. Ivankina, and W. Angole, "Using the RITA threats ontology to guide requirements elicitation: An empirical experiment in the banking sector," in *Proc. 1st Int. Workshop Manag. Requirements Knowl.*, Munich, Germany, Sep. 2008, pp. 11–15.
[20] L. A. Hermoye, A. V. Lamsweerde, and D. E. Perry. (2014). *A Reuse-Based Approach to Security Requirements Engineering*. [Online]. Available: http://users.ece.utexas.edu/~perry/work/papers/060908-LH-reuse.pdf
[21] B. Hamid, "A model-driven approach for developing a model repository: Methodology and tool support," *Future Gener. Comput. Syst.*, vol. 68, pp. 473–490, Mar. 2017.
[22] B. Hamid and J. Perez, "Supporting pattern-based dependability engineering via model-driven development: Approach, tool-support and empirical validation," *J. Syst. Softw.*, vol. 122, pp. 239–273, Dec. 2016.
[23] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating Security and Systems Engineering*, West Sussex, U.K.: Wiley, 2006.
[24] J. M. Bradshaw, A. Uszok, M. Breedy, L. Bunch, T. C. Eskridge, P. J. Feltovich, M. Johnson, J. Lott, and M. Vignati, "The KAoS policy services framework," in *Proc. 8th Cyber Secur. Inf. Intell. Res. Workshop*, Oak Ridge, TN, USA, 2013, pp. 1–4.
[25] D. Alrajeh, A. Cailliau, and A. van Lamsweerde, "Adapting requirements models to varying environments," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, Seoul, South Korea, Jun. 2020, pp. 50–61.
[26] K. Supaporn, N. Prompoon, and T. Rojkangsadan, "An approach: Constructing the grammar from security pattern," in *Proc. 4th Int. Joint Conf. Comput. Sci. Softw. Eng.*, Hong Kong, 2007, pp. 1–8.
[27] H. Mouratidis and P. Giorgini, "Secure Tropos: A security-oriented extension of the Tropos methodology," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 17, no. 2, pp. 285–309, Apr. 2007.
[28] P. Siswahyudi, T. A. Kurniawan, and V. Sugiarto, "Agent-oriented methodologies comparison: A literature review," *Adv. Sci. Lett.*, vol. 24, no. 11, pp. 8710–8716, Nov. 2018.
[29] M. Riaz and L. Williams, "Security requirements patterns: Understanding the science behind the art of pattern writing," in *Proc. 2nd IEEE Int. Workshop Requirements Patterns*, Chicago, IL, USA, Sep. 2012, pp. 29–34.
[30] R. Mazo and C. Feltus, "Framework for engineering complex security requirements patterns," in *Proc. 6th Int. Conf. IT Converg. Secur. (ICITCS)*, Prague, Czech Republic, Sep. 2016, pp. 1–5.
[31] L. P. Gonçalves and A. R. da Silva, "Towards a catalogue of reusable security requirements, risks and vulnerabilities," in *Proc. 27th Int. Conf. Inf. Syst. Develop.*, Lund, Sweden, 2018, pp. 1–12.
[32] L. Goncalves and A. R. da Silva, "A catalogue of reusable security concerns: Focus on privacy threats," in *Proc. IEEE 20th Conf. Bus. Informat. (CBI)*, Vienna, Austria, Jul. 2018, pp. 52–61.
[33] D. de Almeida Ferreira and A. R. da Silva, "RSLingo: An information extraction approach toward formal requirements specifications," in *Proc. 2nd IEEE Int. Workshop Model-Driven Requirements Eng.*, Lisbon, Portugal, Sep. 2012, pp. 39–48.
[34] A. R. D. Silva and D. Savić, "Linguistic patterns and linguistic styles for requirements specification: Focus on data entities," *Appl. Sci.*, vol. 11, no. 9, pp. 4119–4152, 2021.
[35] S. Salva and L. Regainia, "Using data integration to help design more secure applications," in *Proc. Int. Conf. Risks Secur. Internet Syst.*, Dinard, France, 2017, pp. 83–98.
[36] Q. Ramadan, D. Strüber, M. Salnitri, J. Jürjens, V. Riediger, and S. Staab, "A semi-automated BPMN-based framework for detecting conflicts between security, data-minimization, and fairness requirements," *Softw. Syst. Model.*, vol. 19, no. 5, pp. 1191–1227, Sep. 2020.
[37] F. Obeid and P. Dhaussy, "Formal verification of security pattern composition: Application to SCADA," *Comput. Informat.*, vol. 38, no. 5, pp. 1149–1180, 2019.
[38] R. Wirtz and M. Heisel, "Managing security risks: Template-based specification of controls," in *Proc. 24th Eur. Conf. Pattern Lang. Programs*, Irsee, Germany, 2019, pp. 1–13.
[39] G. Müller, "Systems engineering research methods," in *Proc. Conf. Syst. Eng. Res.*, Atlanta, GA, USA, 2013, pp. 1–10.
[40] A. Tripathy and S. K. Rath, "Application of natural language processing in object oriented software development," in *Proc. Int. Conf. Recent Trends Inf. Technol.*, Chennai, India, Apr. 2014, pp. 1–7.
[41] Leximancer, "Leximancer," Brisbane, QLD, Australia, 2019. [Online]. Available: https://info.leximancer.com/
[42] T. Davison, A. Gregory, T. Parker, and E. Waller, "Software requirements specification (SRS) iMedLife personal medical record application for the iPhone," Comput. Sci. Eng., Michigan State Univ., East Lansing, MI, USA, 2009.
[43] J. Ciliberti, E. Hoyt, B. Mack, J. Lewis, and J. Zalewski, "Florida Gulf Coast University digital hospital and medical information system," Comput. Inf. Syst. Program College Bus., Florida Gulf Coast Univ., Fort Myers, FL, USA, Tech. Rep., 2015.
[44] R. Shah, "Analyzing and dissecting Android applications for security defects and vulnerabilities," Blueinfy, Ahmadabad, India, Tech. Rep., 2011.
[45] R. Anderson, *Security Engineering*. Hoboken, NJ, USA: Wiley, 2008.
[46] C. Wysopal, L. Nelson, D. D. Zovi, and E. Dustin, *The Art of Software Security Testing—Identifying Software Security Flaws*. London, U.K.: Pearson, 2006.
[47] R. Subramanian, *Computer Security, Privacy, and Policies: Current Issues, Challenges, and Solutions*. Calgary, AB, Canada: Idea Group, 2008.
[48] C. Jones, *Software Engineering Best Practices: Lessons From Successful Projects in the Top Companies*. New York, NY, USA: McGraw-Hill, 2009.
[49] M. S. Merkow and L. Raghavan, *Secure and Resilient Software: Requirements, Test Cases, and Testing Methods*. Boca Raton, FL, USA: CRC Press, 2011.
[50] A. Adepetu, K. A. Ahmed, Y. Al Abd, A. A. Zaabi, and D. Svetinovic, "CrowdREquire: A requirements engineering crowdsourcing platform," in *Proc. AAAI Spring Symp., Wisdom Crowd*, Stanford, CA, USA, 2012, pp. 1–6.
[51] S. Agrawal and D. Vieira, "A survey on Internet of Things," *Abakós*, vol. 1, no. 2, pp. 78–95, May 2013.
[52] I. Alqassem and D. Svetinovic, "A taxonomy of security and privacy requirements for the Internet of Things (IoT)," in *Proc. IEEE Int. Conf. Ind. Eng. Eng. Manage.*, Selangor, Malaysia, Dec. 2014, pp. 1244–1248.