An Ontology-based Approach for Automatic Specification, Verification, and Validation of Software Security Requirements: Preliminary Results

Dimitrios Tsoukalas, Miltiadis Siavvas, Maria Mathioudaki, Dionysios Kehagias
Centre for Research and Technology Hellas, Thessaloniki, Greece
tsoukj@iti.gr, siavvasm@iti.gr, mariamathi@iti.gr, diok@iti.gr

***Abstract*—Critical software vulnerabilities are often caused by incorrect, vague, or missing security requirements. Hence, there is a strong need in the software engineering community for tools that facilitate software engineers in eliciting and evaluating security requirements. Although several methods have been proposed for specifying, verifying, and validating security requirements, they require a lot of manual effort by requirement engineers, which hinders their practicality. To this end, we introduce a software security requirements specification mechanism, able to automatically identify the main concepts of a given set of security requirements expressed in natural language. Our mechanism applies syntactic and semantic analysis in order to transform requirements into appropriately structured ontology objects. We also propose a software security requirements verification and validation mechanism, which compares a given security requirement to a curated list of well-defined security requirements based on similarity checks, identifies inconsistencies, and proposes refinements. Both of the proposed mechanisms comprise standalone tools, implemented in the form of web services. The capabilities of the proposed mechanisms are demonstrated through a set of test cases.**

***Index Terms*—software security, software security requirements, requirements engineering**

I. Introduction

Nowadays, software applications comprise a critical element of our society, as important daily activities, such as communication, transportation, etc., highly depend on software. The high inter-connectivity of modern software systems along with their increasing accessibility through the Internet, renders them potential targets of malicious attacks [1], [2]. In particular, security weaknesses (i.e., vulnerabilities) that reside in software systems can be exploited by malicious parties in order to compromise the system and infringe critical security requirements, including Confidentiality, Integrity, and Availability. Hence, identifying and eliminating security weaknesses early enough in the Software Development Lifecycle (SDLC), prior to the release of the software systems, is crucial for engineering secure and reliable software.

A software vulnerability is defined as a weakness in the specification, development, or configuration of software, so that its exploitation can violate a security policy [3]. Most of the software vulnerabilities stem from a small number of common programming errors that are introduced by the developers during the implementation phase, mostly as a result of their lack of security expertise [4], [5]. However, a significant part of the security vulnerabilities that reside in software projects are also introduced during the design and requirement phases of the SDLC, mainly because of the incorrect, vague, or missing security requirements introduced in the design phase [1], [6]. Those vulnerabilities that are introduced in the requirement elicitation and design phases of the SDLC, manifest themselves as actual vulnerabilities in the source code, unless they are detected and mitigated promptly. Moreover, these types of vulnerabilities are more difficult and expensive to be fixed, as they require the software engineers to revisit the original requirements and make changes to the design and source code of the underlying system in order to be in line with the new specifications. This results in an increasing need for extensive refactorings to the existing code base, but also in potential corruptions of previously secure code, which in turn introduces new bugs and vulnerabilities [7], [8].

Hence, there is an imperative need for methods and tools that will facilitate and clarify the definition of software security requirements, as well as their verification and validation, in order to help software engineers better define the security requirements of a given software project. Recently, several methods for facilitating the correct and clear specification of security requirements have been proposed in the software engineering literature, focusing mainly on introducing theoretical methodologies, reusable security requirement templates, and formal languages [9]. However, very limited work has been dedicated to facilitating the automatic elicitation and formal specification of security requirements from requirement documents expressed in natural language, which is the format in which requirements are usually expressed in practice [10], [11]. In addition to this, although some initial attempts have been noticed with respect to evaluating the correctness and the completeness of the defined security requirements [12], [13], very few of them have managed to demonstrate sufficient results especially with an enhanced level of automation [14].

In order to address the aforementioned issues, this paper introduces a set of techniques for formally specifying software security requirements, verifying and validating their correctness and completeness, and providing recommendations for potential refinements that are deemed necessary in order to avoid future security issues. More specifically, this paper introduces a Software Security Requirements Specification (SSRS) mechanism, able to automatically identify the main concepts of a given set of security requirements expressed in natural language by applying syntactic and semantic analysis, and subsequently turn them into more well-structured ontology objects. It also introduces a Software Security Requirements Verification and Validation (SSRVV) mechanism, which compares a given security requirement to a curated list of well-defined security requirements based on similarity checks, identifies inconsistencies, and proposes appropriate refinements.

The work introduced in this paper comprises preliminary results in the context of the ongoing research project IoTAC[1], funded by the European Commission (EC) under the Horizon 2020 Work Programme. In particular, this paper describes two standalone tools of the IoTAC Security-by-Design (SSD) Platform, developed as individual web services, allowing us to demonstrate their capabilities through a set of test cases.

The rest of this paper is structured as follows: Section 2 presents the related work in the field of software security requirements engineering, and more specifically in requirements specification and verification & validation areas. Section 3 describes in sufficient detail our work regarding the design and development of SSRS and SSRVV mechanisms and tools, accompanied by preliminary results of their application on simple test cases. Finally, Section 4 concludes the paper and discusses future work.

II. Related Work

Security Requirements Engineering (SRE) is a sub-field of Software Engineering that is based on the concept that security requirements should be analyzed early in the SDLC [2], [9], [15]. In order to produce secure software, security requirements should be identified and correctly specified from the early stage of requirement elicitation phase, along with the functional requirements of the system [9], [15]. Incorrect, vague, and incomplete lists of security requirements lead to the introduction of critical vulnerabilities in software systems [1], [16], caused mainly by the fact that requirements engineers often lack security expertise. Hence, this leads to a need for security experts to be trained for security requirements elicitation tasks, which in turn results in an increased effort and cost [15]. Besides, the late identification of these inconsistencies in the security requirements leads to significant rework that needs to be performed on the source code, which requires additional effort and increases the risk of introducing new issues [7], [8]. Hence, there is a strong need for mechanisms and tools that will assist requirement engineers verify and validate the correctness and completeness of the defined security requirements, and specify them in a clear way, encapsulating the required security expertise [2].

[1]https://iotac.eu/

In the following, we present the state of the art in the field of SRE, specifically focusing on the mechanisms and techniques that have been proposed over the years for facilitating the specification, verification, and validation of software security requirements. Emphasis is given on the challenges and open issues of the existing approaches that our proposed solutions are trying to address.

A. Software Security Requirements Specification

As already mentioned, the correct specification of security requirements is important for avoiding potential inconsistencies in the defined security requirements, and, in turn, the introduction of potential vulnerabilities in later stages of the development. To this end, several approaches, methods, and techniques for requirements specification have been proposed over the years. Most of the existing research endeavors have focused on creating specialized languages, which can then be easily translated to models. For instance, in a study by Kaindl et al. [17], a constrained language named Requirements Specification Language is introduced for this purpose. Another notable example is Tropos [18], a requirements-driven methodology based on the notions of actors and goals of the i* modelling framework [19]. In addition, Majumdar et al. [20] introduce Adv-EARS, a framework for expressing requirements in a formal syntax, which is parsed by a grammar analyser to identify potential use cases and actors.

Despite their benefits in enabling the formal specification of security requirements, these approaches are hindered by the fact that they introduce additional effort to the engineers workload. More specifically, requirement engineers need to be trained in order to learn these custom languages and also work on manually expressing the requirements into the required format, which is a time consuming and tedious task. In practice, security requirements are expressed in natural language (free text), since they need to be easily understandable even by stakeholders with little or no technical knowledge, as well as easier to define. Hence, mechanisms for turning actual security requirements into formal expressions (i.e., languages, templates, etc.) are necessary.

In order to facilitate the automatic elicitation of security requirements from natural text, several approaches have been proposed, mainly focusing on using Natural Language Processing (NLP) techniques. For instance, in [10], the authors employ NLP models in order to identify implied security requirements that can be found in functional requirement documents, and consequently propose templates that can be used for manually defining these security requirements. Similarly, in [21], the authors employ NLP to classify a requirement into its grammatical constructing elements, and then reconstruct it based on a particular schema (i.e. "Requirement <- Subject + Verb + Target + [Way]"). Furthermore, in [11], the authors introduce an approach for automatic extraction of a structural model from semi-structured software requirements using NLP tools, whereas in [22] the authors propose the R-TOOL, which analyses texts of software requirements to generate actors, use cases, classes, and relationships between the classes, leading to the generation of class diagrams. However, no concrete techniques have been proposed to enable the automatic elicitation of formal security requirement specifications from security requirements expressed in free text.

On the other hand, promising techniques have been applied for the elicitation of Functional Requirements. For instance, Roth et al. [23] proposed converting requirements written in natural language to formal semantic representations for analysis and reuse purposes. This idea is further developed in [24] that uses semantic role labelling to automate mapping of functional requirements and use case diagrams to formal representations expressed as ontology concepts. The authors in [25] and [26] extend previous work by using ontology concepts to capture also the dynamic view of software projects (e.g., activity diagrams). Although these concepts could be extended towards the security realm, in order to facilitate the automatic specification of security requirements, to the best of our knowledge no work has been conducted towards this direction so far.

B. Software Security Requirements Verification & Validation

Verifying and validating security requirements is critical for identifying and fixing potential inconsistencies in the security requirements definition. Regarding Software Security Requirements Verification & Validation (V&V), a relevant work by Besrour [27] proposes a way to measure security in requirements engineering through a security checklist that contains a set of security questions that have to be answered properly. Then a traceability matrix is used for risk assessment. In another study by Shaaban et al. [13], the authors introduce a method for verifying and validating the selected security requirements and calculating a security score. They also devise a set of rules for selecting additional security requirements from a security requirements ontology.

A different approach is introduced in [28], where the authors propose a software requirements V&V activity that checks whether the allocation of system requirements to software is appropriate and correct, and how well the software requirements have been specified. Results show that requirements verification should include an examination of documentation produced earlier in system development and input to the V&V activity could be documents either in natural language or in formal mathematical languages. Besides, software requirements verification should ensure fidelity among the forms of representation, such as conformity to a pre-decided template. Finally, Atoum [12] proposes a scalable operational framework to learn, predict, and recognize requirements defects using semantic similarity models and the Integration Functional Definition methods. The proposed framework facilitates the validation process and increases the productivity of software engineers online with customer needs.

However, all of the existing approaches require a lot of manual effort by the requirements engineers, hindering, in that way, their practicality. Actually, to the best of our knowledge, none of the existing mechanisms automatically verifies and validates the correctness and completeness of security requirements and proposes refinement recommendations.

III. Methodology

The Requirements Specification and Verification & Validation (V&V) approaches that are introduced in this paper comprise preliminary results produced in the context of the ongoing IoTAC project. One of the main goals of the IoTAC project is to facilitate the development of more resilient IoT applications through monitoring and evaluating their security. This will be achieved through the IoTAC Software Security-by-Design (SSD) Platform[2] that provides solutions for assessing, improving, and certifying the security level of IoT software applications throughout the overall SDLC. More specifically, the SSD Platform provides tools for better specifying the security requirements of a given IoT software application, assessing its adherence to the originally imposed requirements, detecting software vulnerabilities that reside in the source code, and identifying areas in the source code, namely security hotspots, that require specific care from a security viewpoint. It will also provide solutions for certifying the security levels of the analysed software by taking into account conformance to International Standards. Fig. 1 provides a high-level overview of the IoTAC SSD Platform.

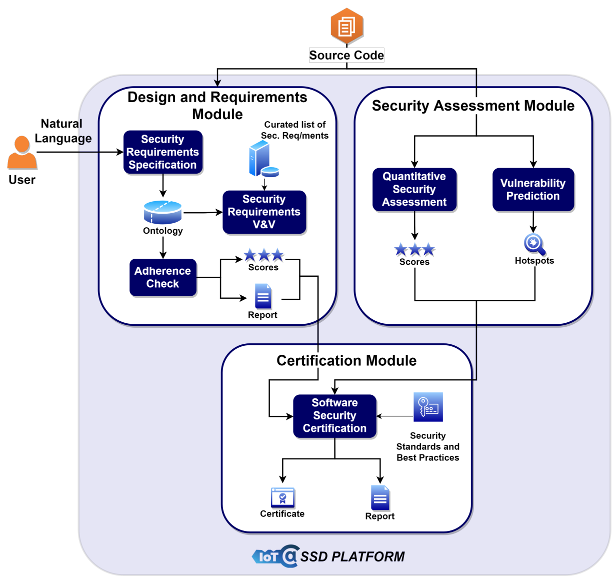

Source Code
Design and Requirements Module
Natural Language
User
Security Requirements Specification
Curated list of Sec. Req/ments
Ontology
Security Requirements V&V
Adherence Check
Scores
Report
Security Assessment Module
Quantitative Security Assessment
Vulnerability Prediction
Scores
Hotspots
Certification Module
Software Security Certification
Security Standards and Best Practices
Certificate
Report
IoT SSD PLATFORM

Fig. 1. The high level overview of the IoTAC Software Security-by-Design (SSD) Platform

The ongoing work presented in this paper is related to the Design and Requirements module, whose purpose is to provide solutions for monitoring (i.e., measuring) and improving (i.e., optimizing) the security level of a software application during the Requirements and Design phases of the overall SDLC. Although not directly related to the Development (i.e., Coding) phase, the contribution of this module to the overall security

[2]https://iotac.eu/the-iotac-software-security-by-design-ssd-platform-concept-and-preliminary-overview/

improvement is considered equally important, since a large percentage of vulnerabilities that reside in the source code are introduced due to poor design decisions, as well as by incorrect, vague, and/or missing security requirements. To this end, as shown in Fig. 1, the Design and Requirements module will provide three core functionalities, which will be implemented in the form of individual components, namely the Software Security Requirements Specification (SSRS) component, the Software Security Requirements V&V (SSRVV) component, and the Software Security Requirements Adherence Check (SSRAC) component, respectively.

In the remaining section, the first two individual components of the Design and Requirements module are thoroughly described, mainly focusing on the methodology and overall approach that was adopted for their conceptualization and implementation as standalone tools.

A. Software Security Requirements Specification

The SSRS component, serving as an entry point to the Design and Requirements module, aims to facilitate the specification of the security requirements of a given software product. More specifically, it is responsible for enabling the user to define a set of desired security requirements in natural language and then identifying the main concepts and underlying semantics of the specified requirements automatically, in order to enable further processing. For the purpose of automatic transformation of user-specified requirements in machine-readable form, syntactic and semantic analysis techniques, but also ontology-oriented programming tools, were considered and properly configured to parse the requirements and support ontology manipulation, respectively.

As already mentioned in Section 2, recent research work [29] on requirements specification has introduced the concept of automatic transformation of user requirements to system specifications by employing NLP techniques and semantics. After deriving the requirements specifications, the goal is to store them as formal representations in ontologies. Since the ontologies are inherently organized in specific classes, the objective is to map the different aspects (i.e., concepts) of the requirements to these classes. The SSRS component that is described in this section, builds upon this concept and aspires to develop mechanisms able to automatically parse user-specified security requirements using NLP, extract their main concepts in machine-readable form and finally store them into an ontology. As the description of the aforementioned process might sound complicated, the remaining section will go through each step in detail.

1) Overview of the Ontology Concept: In what follows, an outline of the ontology schema that will be used for requirements specification is provided, emphasizing on the different aspects (i.e., concepts) that need to be extracted from the software requirements. As in the case of the main structural concepts (Subject-Verb-Object) that comprise a sentence in natural language, the schema of the ontology that is dedicated to store requirement aspects should evolve around the concept that "an Actor does some Action on some Object". Ontologies can also host relationships with complex connections, such as 'is-a'-, 'has-a'- and 'use-a'-relationships, but also inverse relationships. In our case, as mentioned before, software security requirements are formed as sentences that describe relations between actors, actions, objects, etc. These concepts can be interconnected with various types of relations, such as 'has-action', 'is-actor-of', 'acts-on', as well as their inverses.

In this context, we have designed an expressive ontology[3] schema that tries to capture as many aspects of the static view of a security requirement as possible. The class hierarchy of this ontology is defined as follows: anything that enters the ontology is *Requirement*, which is further classified into a *Project*, *Priority*, *Security Characteristic*, *Action*, *Actor*, *Object* and *Property*. The Requirement class is used in order to store the actual requirement text and link any requirement property to it, so that any instance can be traced back to the initial requirement. In addition, the *Project* subclass is used to link any requirement to the software project under consideration, so that one can easily recall all requirements belonging to a specific project. *Priority* and *Security Characteristic* subclasses refer to the importance of the requirement (e.g., must, should, could) and the security category (e.g., Confidentiality, Integrity, etc.) that it tries to cover, based on a classification proposed by the ISO 25010 [30]. Finally, *Action*, *Actor*, *Object* and *Property* subclasses are the key types of concepts found in any requirement and refer to the linguistic properties of its sentence (Subject-Verb-Object). *Action* basically describes an operation performed by an *Actor* on some *Object*, while *Property* includes all modifiers of objects that assign some property to the object involved. As an example, let us consider the security requirement *"The system must authenticate external users"*, having "*Authenticity*" as its security characteristic. The priority is "*must*", the actor is the "*system*", the action is "*authenticate*" and the object is "*users*", while "*external*" is the property of the object. It should be mentioned here that a requirement can have more than one Action-Actor-Object triplet.

2) Extracting Ontology Instances from Security Requirements: As soon as the ontology schema has been defined, the ontology will be used to store transformed security requirements. In order to assign the requirements to the ontology however, they first need to be parsed into requirement concepts. The parsing process is in fact the transformation (mapping) of the requirements expressed in natural language[4] into basic requirement concepts (i.e., Actors, Actions, Objects, etc.), which are then stored as instances of the corresponding ontology class. Within the context of the SSRS component, the decomposition of a requirement into its concepts is performed using a process that consists of two main steps: the *Syntactic Analysis* and the *Semantic Analysis*. A high-level overview of the architecture of the SSRS component is shown in Fig. 2.

As shown in Fig. 2, the pipeline architecture of the SSRS component starts with the Syntactic Analysis sub-component.

[3]We use the Web Ontology Language (OWL) for ontology representation, since it is a widely used standard within research and industry

[4]For now our methodology supports only the English language

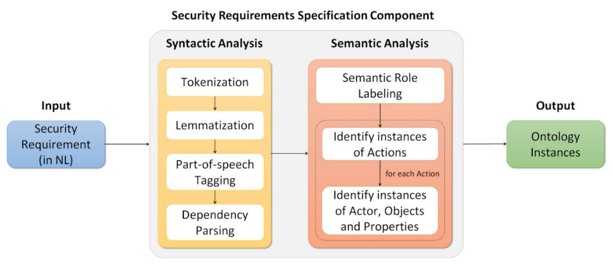


Fig. 2. Pipeline architecture of the SSRS component

In brief, this sub-component takes as input a requirement expressed in natural language, tokenizes it into words, and then identifies the grammatical category of each word (e.g. noun, verb, etc.), as well as the grammatical relations that hold between the words (e.g., subject-verb-object etc.). In order to achieve that, there are some key steps that are followed when given a requirement sentence as an input:

1) *Tokenization*: which is the separation of the sentence into word tokens
2) *Lemmatization*: which determines the uninflected base form of each word
3) *Part-of-speech tagging*: which identifies the grammatical category of each word
4) *Dependency parsing*: which identifies the grammatical relations between words

For each step of the syntactic analysis process described above, the SSRS component relies on third-party components and pre-trained models from Mate Tools[5] [31], [32] library. Mate Tools is a bundle of widely known syntactic analysis tools that use a state-of-the-art dependency parser [33] and semantic role labeller [34]. These tools are language independent, provide a very high accuracy and are fast.

Once the Syntactic Analysis sub-component finishes its analysis, the next step in the SSRS architecture pipeline is the Semantic Analysis sub-component. As shown in Fig. 2, this sub-component receives as input the syntactic analysis results, i.e., the part-of-speech tagging and dependency parsing, and combines this information with a semantic role labelling process in order to identify instances of the main requirement concepts (e.g., Actions, Actors, Objects, etc.) and relations. Briefly, the semantic analysis consists of the following steps:

1) Semantic Role Labelling
2) Identification of the main requirement concepts:
    a) Identification of the Priority of the requirement
    b) Identification of instances of requirement Actions
    c) For every Action identified in step b., identification of instances of the rest of the important concepts (i.e., Actor, Object and Property)

As stated above, the semantic analyser comprises two main steps, one for semantic role labelling, and the other for identifying instances of the main requirement concepts and the relationships among them (indicated within the dotted box in Fig. 2). In brief, semantic role labelling is a process that assigns labels to words or phrases that indicate their semantic role in the sentence. While Mate Tools (used within the Syntactic Analysis sub-component) already provide a typical implementation of a semantic role labeller, in our case it is not enough, since we need to label words based on requirement concepts, rather than the typical thematic relations. Therefore, while we consider the output provided by Mate Tools, we further enhance the semantic role labelling process by adding support for arbitrary label types that represent the main requirement concepts (e.g., Priority, Actions, Actors, etc.). To do so, we have developed a set of sophisticated rules that, besides the grammatical category (e.g. noun, verb, etc.) and grammatical relations that hold between the words (e.g., subject-verb-object etc.), take into account also other lexical semantic and syntactic properties, such as the relation of a word to its parent, the parent's part-of-speech, the dependencies among each word, the order of words within the sentence, etc. to accurately determine the role (i.e., concept) that a word may hold within a security requirement.

Therefore, the semantic analysis is carried out in three steps. Step 1 involves the identification of the Priority and all instances of Actions within a requirement sentence. Then, for each identified Action in step 1, step 2 involves the identification of the Actor and Objects that are related to this specific Action. Finally, for each identified Object in step 2, step 3 involves the identification of all Properties that are related to this specific Object. An example of the results produced by the SSRS component is presented in Section IV.

3) Constructing the Security Requirements Knowledge Base: The syntactic and semantic analysis processes introduced earlier, besides parsing an arbitrary user-specified security requirement, were also used for building (i.e., populating) the Security Requirements Knowledge Base of the SSD Platform. Security Requirements Knowledge Base is an ontology containing a carefully curated set of software security requirements that, as shown in Fig. 1, will serve as a reference point for requirements V&V tasks (more on that in the next section). Within the context of this work, this ontology is continuously populated with security requirements taken from repositories of well-defined security requirements covering various categories (e.g., Confidentiality, Integrity, etc.).

B. Software Security Requirements Verification & Validation

The purpose of the SSRVV component is to evaluate the correctness and completeness of the software security requirements defined by the user and provide recommendations regarding their improvement. The overall approach of the SSRVV component is presented in Fig. 3.

As shown in Fig. 3, the V&V process starts with a set of machine-readable security requirements given as input to the component. This input is actually the result of the previously described SSRS component, where the main concepts of any requirement provided by the user are extracted as ontology instances. Next, the SSRVV component assesses the level of correctness and completeness of each of the requirements

[5]https://code.google.com/archive/p/mate-tools/

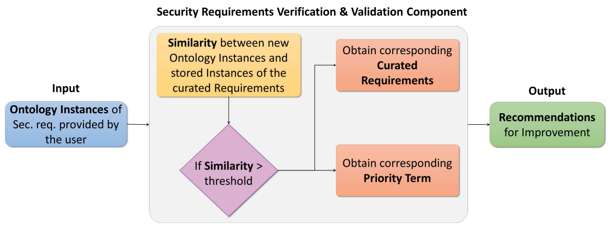

Security Requirements Verification & Validation Component
Input
Ontology Instances of Sec. req. provided by the user
Similarity between new Ontology Instances and stored Instances of the curated Requirements
If Similarity > threshold
Obtain corresponding Curated Requirements
Obtain corresponding Priority Term
Output
Recommendations for Improvement

Fig. 3. Pipeline architecture of the SSRVV component

provided as input, based on a curated security requirements set (i.e., the Security Requirements Knowledge base), and suggests refinements to the user as output. For this purpose, popular NLP toolkits (e.g., WordNet with NLTK[6]) are considered in order to perform similarity checks between the user-defined requirements and the curated security requirements set. If the similarity score is found to be higher than a specific threshold, then recommendations for improvements are presented to the user in the form of:

- *similar or additional requirements from the Security Requirements Knowledge base* that could be considered by the user in order to replace or complement the initial set of user-defined requirements, and
- *suggestions of alternative Priority Terms* that better reflect the severity level of the initial set of user-defined requirements.

The above recommendations are expected to lead to a final set of security requirements that are more correct and homogeneous, reducing in that way the probability of introducing vulnerabilities during the Design phase of the SDLC.

1) Software Security Requirements Recommendations: As mentioned previously, the input to the SSRVV component is actually the result of the SSRS component. More specifically, the SSRVV component receives the instances of the requirement concepts (i.e., Priority, Actors, Actions, Objects and Properties) for each user-defined requirement sentence, as identified during the syntactic and semantic analysis steps. These ontology instances are then "compared" to the corresponding instances of each curated requirement stored into the Security Requirements Knowledge base, using the Wordnet[7] library provided by NLTK corpus to employ word similarity checks. More specifically, we make use of the *Path Similarity* method, which returns a score (in the range 0 to 1) denoting how similar two word senses are, based on the shortest path that connects the senses in the is-a (hypernym/hyponym) taxonomy. Subsequently, if "significant" similarity (i.e., more than a predefined threshold) is found between them, then the corresponding requirement of the curated list is presented to the user as a recommendation. A user-defined requirement is considered "similar" to a curated one if: i) all of its three key concepts (i.e., Actor, Action, Object) have similarity scores higher than 0.7 when compared to the respective three concepts of the curated requirement, or ii) both of the concepts in one of <Actions-Objects>, <Actors-Objects>, or <Actors-Actions> tuples have similarity scores higher than 0.9 when compared to the respective concepts of the curated requirement[8]. In this way, the user is given the opportunity to consider replacing his/her initially loosely specified security requirement with a curated requirement that will have similar or equal meaning to what he/she is trying to express, but at the same time will be provided by experts in the domain so it is more likely to be clearer and more complete.

Apart from providing recommendations on potential replacement of user-specified requirements with alternatives that are better in terms of syntax and meaningfulness, this component serves another purpose, which is to identify any curated requirement that may complement the initial user-specified requirements. To this end, it constructs a more complete set of security requirements for a particular goal. Based again on the similarity between the instances of the requirement concepts, as well as on the "Security Characteristic", which denotes the security category that a requirement belongs to (e.g., Confidentiality, Authentication, etc.), the user is presented with a homogeneous group of requirements that either belong to the same security category, or to categories that are closely related to each other (e.g., that of Authentication and Authorization).

2) Priority Term Recommendation: Another aspect of the SSRVV component is the suggestions of alternative Priority terms that better reflect the severity level of the initial user-defined requirements, based again on the curated requirements list. By a Priority term, we basically refer to the modal verb that is used in a requirement ("must", "should", "could", "won't", etc.) and denotes the significance and urgency that a particular requirement has. We follow the MoSCoW approach [35], which is a commonly used prioritization method to quantify the priority of a requirement so that a mutual understanding among stakeholders is ensured. For this purpose, similarly to the "Requirements Recommendations" step, we employ Wordnet to perform a similarity check between a user-defined requirement and each curated requirement stored into the Security Requirements Knowledge base. Subsequently, we check whether the similarity value is above a predefined threshold and if so, then the priority term of the corresponding curated requirement is presented to the user as a suggestion.

IV. Technical Implementation and Test Cases

A. Software Security Requirements Specification Component

Based on the modelling effort described in Section III-A, we have developed an operational version of the SSRS component in the form of a web service. This web service allows the individual and remote invocation of the SSRS functionalities, but it also important for the future integration of this component with the final IoTAC SSD Platform, which will be web-based. The web service provided by the SSRS component, namely *SemanticsToReqRelations*, is responsible for transforming a

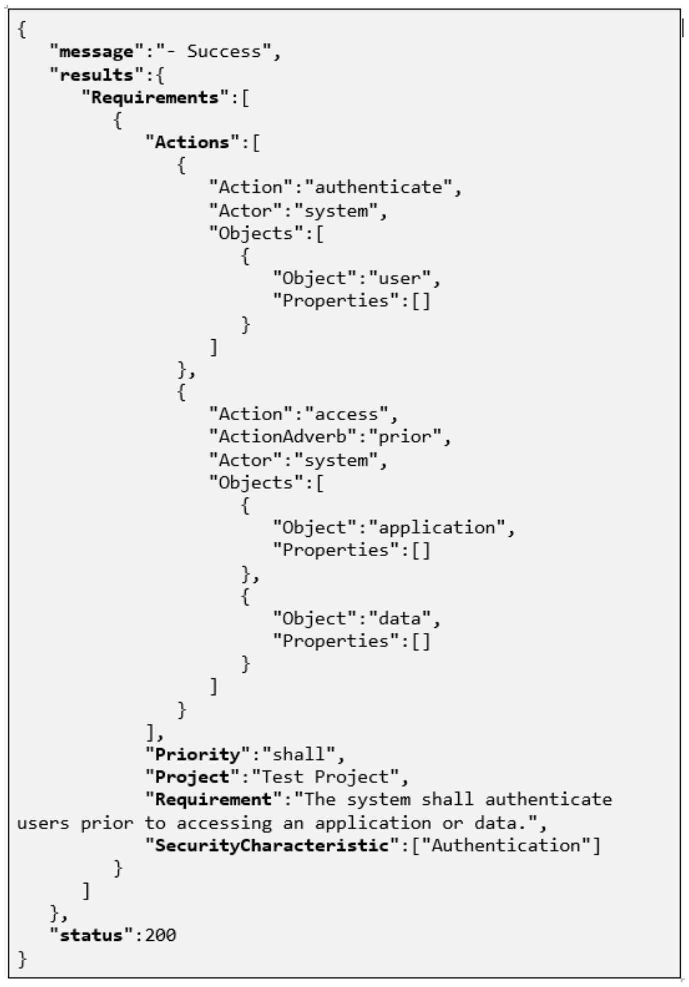
[6]https://www.nltk.org/

[7]https://wordnet.princeton.edu/

[8]The above thresholds are likely to be updated in the future based on large-scale evaluations and testing. It should be also noted, that the final mechanism will be highly configurable allowing the users to define their own thresholds.

set of security requirements into machine-readable format and storing them into a domain ontology respectively.

The SemanticsToReqRelations web service integrates the syntactic and semantic analysis processes described in Section III-A2 to automatically map software requirements written in natural language to requirement concepts (Action, Actor, Properties, etc.) and relations (is-actor-of, acts-on, has-property, etc.). This is achieved through a dedicated API exposed by the RESTful web server, which allows the user to perform a simple HTTP POST request to the web service. More specifically, this service receives as input a list of user-defined security requirements expressed in natural language, along with their respective Security Characteristic category (e.g., Confidentiality, Integrity, etc.). As an example, let us consider the following requirement provided as input to the service: "*The system shall authenticate users prior to accessing an application or data.*". The output of SemanticsToReqRelations service is a JSON object containing the identified requirement concepts and their relations, as illustrated in Fig. 4.



```
{
   "message":"- Success",
   "results":{
      "Requirements":[
         {
            "Actions":[
               {
                  "Action":"authenticate",
                  "Actor":"system",
                  "Objects":[
                     {
                        "Object":"user",
                        "Properties":[]
                     }
                  ]
               },
               {
                  "Action":"access",
                  "ActionAdverb":"prior",
                  "Actor":"system",
                  "Objects":[
                     {
                        "Object":"application",
                        "Properties":[]
                     },
                     {
                        "Object":"data",
                        "Properties":[]
                     }
                  ]
               }
            ],
            "Priority":"shall",
            "Project":"Test Project",
            "Requirement":"The system shall authenticate
users prior to accessing an application or data.",
            "SecurityCharacteristic":["Authentication"]
         }
      ]
   },
   "status":200
}
```

Fig. 4. An example response of the SemanticsToReqRelations web service

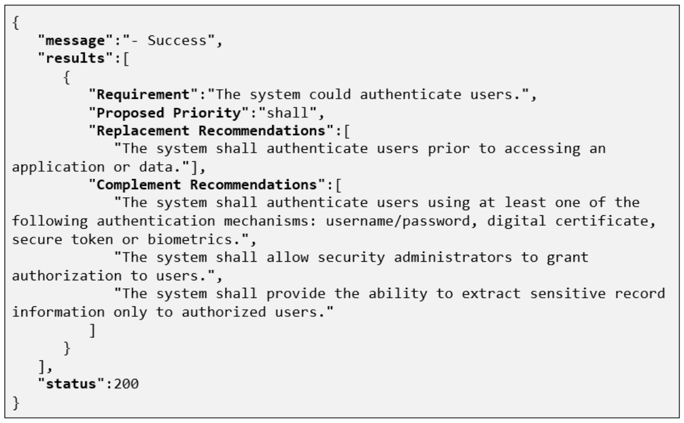
As shown in the JSON fragment above, the SSRS implementation first identifies all instances of Actions within a requirement sentence. Therefore, each entry of the "Requirements" list further contains an "Actions" list. Then, for each identified Action it proceeds with the identification of the "Actor" and "Objects" that are related to this specific Action. Finally, for each identified Object, it further identifies all Properties that are related to this specific Object.

B. Software Security Requirements Verification & Validation Component

Based on the processes described in Section III-A3, we have produced an operational, although still in a preliminary stage, version of the SSRVV component in the form of a web service, namely *RecommendedReqImprovements*, which is responsible for providing the user with useful and meaningful suggestions for the improvement of the security requirements.

The whole process consists of the following principal steps: First, the service receives the output of the SSRS component, i.e., the security requirements that the user has defined during the Requirements Specification phase, transformed into machine-readable format. This machine-readable format consists of the identified main concepts of the requirements, extracted as ontology instances. The second step involves word similarity checks between the ontology instances of this input and the ontology instances of the curated requirements that have been stored in the Security Requirements Knowledge base. Based on these similarity checks, RecommendedReqImprovements service produces an output that contains recommendations and suggestions for improvement.

As already mentioned, the input to the RecommendedReqImprovements web service is in fact the output of the SemanticsToReqRelations web service (see Fig. 4). The output (i.e., the response) of the RecommendedReqImprovements web service is a JSON object containing recommendations for improvement for each requirement given as input. As an example, let us consider the following loosely defined requirement provided as input to the service: "*The system could authenticate users.*". The output of RecommendedReqImprovements service is a JSON object containing recommendations and suggestions for improvement, as illustrated in Fig. 5.



```
{
   "message":"- Success",
   "results":[
      {
         "Requirement":"The system could authenticate users.",
         "Proposed Priority":"shall",
         "Replacement Recommendations":[
            "The system shall authenticate users prior to accessing an
application or data."],
         "Complement Recommendations":[
            "The system shall authenticate users using at least one of the
following authentication mechanisms: username/password, digital certificate,
secure token or biometrics.",
            "The system shall allow security administrators to grant
authorization to users.",
            "The system shall provide the ability to extract sensitive record
information only to authorized users."
         ]
      }
   ],
   "status":200
}
```

Fig. 5. An example response of the RecommendedReqImprovement web service

As shown in the JSON fragment above, the "results" field contains four entries, namely "Requirement", "Proposed Priority", "Replacement Recommendations", and "Complement Recommendations". The "Requirement" field contains the initial requirement that the user provided in natural language, while the "Proposed Priority" contains a suggestion of an alternative Priority term that better reflects the severity level of the specific user-defined requirement. For the user-defined requirement of our example, the service has proposed the priority "shall" instead of "could". Finally, the "Replacement Recommendations" and "Complement Recommendations" fields contains a list of similar or additional curated requirements fetched from the Security Requirements Knowledge base that could be considered by the user in order to replace or complement the initial set of user-defined requirements respectively. For instance, in this particular case, to improve the initial user-specified security requirement "*The system could authenticate users*", the service returned a similar but more complete requirement that could be considered as alternative, that is, "*The system shall authenticate users prior to accessing an application or data.*". In addition, it also returned a set of security requirements that could complement the initial one, such as "*The system shall authenticate users using at least one of the following authentication mechanisms: username/password, digital certificate, secure token or biometrics.*"

V. Conclusions and Future Work

In this paper, we introduced two new approaches for facilitating the specification, verification, and validation of software security requirements, which comprise preliminary results of the ongoing EC-funded research project IoTAC. More specifically, we introduced a mechanism for Software Security Requirements Specification that receives as input security requirements expressed in natural language, identifies the main semantic concepts of the requirements and formally stores them in a dedicated security requirements ontology. In addition to this, we introduced a Software Security Requirements Verification and Validation mechanism, which compares a given security requirement to a curated list of well-defined security requirements based on similarity checks, identifies inconsistencies, and proposes appropriate refinements. The proposed mechanisms have been implemented as prototype tools in the form of web services and their capabilities were demonstrated through dedicated test cases.

Several directions for future work can be identified. First of all, we are planning to evaluate our proposed mechanisms through a large-scale study by utilizing real security requirements retrieved from the requirement documents of actual software products. In particular, security requirements extracted from real world software will serve as a test-bed for evaluating the accuracy of the Software Security Requirements Specification mechanism in detecting the main semantic concepts of the requirements, as well as for assessing the usefulness of the refinement recommendations produced by the Software Security Requirements Verification and Validation mechanism. This is a manual and tedious process, which will also require the involvement of security experts in order to enhance the reliability of the evaluation results. For this purpose, a formal evaluation process will be defined and employed. The aforementioned evaluation process will also allow us to further refine the semantic role labelling rules and similarity thresholds integrated into the two proposed mechanisms. Furthermore, we plan to enhance the Software Security Requirements Verification and Validation mechanism in order to handle special cases of requirements, such as interconnected, overlapping, or redundant ones, as well as in order to also provide recommendations focusing on the quality of the analyzed requirement itself, along with its similarity of the security requirements knowledge base. Last but not least, we plan to link the proposed mechanisms to a dedicated user interface, facilitating their usage in practice.

Acknowledgment

This work is partially funded by the European Union's Horizon 2020 Research and Innovation Programme through IoTAC project under Grant Agreement No. 952684.

References

[1] G. McGraw, "Software security," *IEEE Security & Privacy*, 2004.

[2] N. M. Mohammed, M. Niazi, M. Alshayeb, and S. Mahmood, "Exploring software security approaches in software development lifecycle: A systematic mapping study," *Computer Standards & Interfaces*, 2017.

[3] I. V. Krsul, *Software vulnerability analysis*. Purdue University, 1998.

[4] M. Howard, D. LeBlanc, and J. Viega, *24 deadly sins of software security: Programming flaws and how to fix them*. McGraw-Hill Education, 2010.

[5] M. Siavvas, D. Kehagias, D. Tzovaras, and E. Gelenbe, "A hierarchical model for quantifying software security based on static analysis alerts and software metrics," *Software Quality Journal*, vol. 29, no. 2, pp. 431–507, 2021.

[6] T. Hovorushchenko, A. Boyarchuk, and O. Pavlova, "Ontology-based intelligent agent for semantic parsing the natural language specifications of software requirements," *International Journal on Information Technologies & Security*, vol. 11, no. 2, 2019.

[7] G. J. Holzmann, "The value of doubt," *IEEE Software*, 2017.

[8] A. Bosu, J. C. Carver, M. Hafiz, P. Hilley, and D. Janni, "Identifying the characteristics of vulnerable code changes: An empirical study," in *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, 2014, pp. 257–268.

[9] M. Ramachandran, "Software security requirements engineering: State of the art," in *International Conference on Global Security, Safety, and Sustainability*. Springer, 2015, pp. 313–322.

[10] M. Riaz, J. King, J. Slankas, and L. Williams, "Hidden in plain sight: Automatically identifying security requirements from natural language artifacts," in *2014 IEEE 22nd international requirements engineering conference (RE)*. IEEE, 2014, pp. 183–192.

[11] R. Sanyal, B. Ghoshal, and others, "Automatic extraction of structural model from semi structured software requirement specification," in *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*. IEEE, 2018, pp. 543–58.

[12] I. Atoum, "A scalable operational framework for requirements validation using semantic and functional models," in *Proceedings of the 2nd International Conference on Software Engineering and Information Management*, 2019, pp. 1–6.

[13] A. Magdy, C. Schmittner, T. Gruber, A. B. Mohamed, G. Quirchmayr, and E. Schikuta, "Ontology-based model for automotive security verification and validation," 2019.

[14] A. M. Shaaban, C. Schmittner, T. Gruber, A. B. Mohamed, G. Quirchmayr, and E. Schikuta, "Ontology-based security requirements framework for current and future vehicles," in *Data Science and Big Data Analytics in Smart Environments*. CRC Press, 2021, pp. 197–217.

[15] D. Mellado, C. Blanco, L. E. Sánchez, and E. Fernández-Medina, "A systematic review of security requirements engineering," *Computer Standards & Interfaces*, 2010.

[16] C. P. Pfleeger and S. L. Pfleeger, *Analyzing computer security: A threat/vulnerability/countermeasure approach*, 2012.

[17] H. Kaindl, M. Smialek, D. Svetinovic, A. Ambroziewicz, J. Bojarski, W. Nowakowski, T. Straszak, H. Schwarz, D. Bildhauer, J. Brogan, and others, "Requirements specification language definition: Defining the redseeds languages," *Public deliverable, ReDSeeDS project*, 2007.

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

[19] E. Yu, "Modeling Strategic Relationships for Process Reengineering." *Social Modeling for Requirements Engineering*, no. 2011.

[20] D. Majumdar, S. Sengupta, A. Kanjilal, and S. Bhattacharya, "Automated Requirements Modelling with Adv-EARS," *International Journal of Information Technology Convergence and Services*, pp. 57–67, 2011.

[21] A. Fatwanto, "Software requirements specification analysis using natural language processing technique," in *2013 International Conference on QiR*. IEEE, 2013, pp. 105–110.

[22] S. Vinay, S. Aithal, and P. Desai, "An NLP based requirements analysis tool," in *International Advance Computing Conference. IEEE*, 2009.

[23] M. Roth, T. Diamantopoulos, E. Klein, and A. Symeonidis, "Software requirements: A new domain for semantic parsers," in *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, 2014, pp. 50–54.

[24] T. Diamantopoulos, M. Roth, A. Symeonidis, and E. Klein, "Software requirements as an application domain for natural language processing," *Language Resources and Evaluation*, vol. 51, no. 2, 2017.

[25] C. Zolotas, T. Diamantopoulos, K. Chatzidimitriou, and A. Symeonidis, "From requirements to source code: a Model-Driven Engineering approach for RESTful services," *Automated Software Engineering*, 2017.

[26] T. Diamantopoulos and A. Symeonidis, "Enhancing requirements reusability through semantic modeling and data mining techniques," *Enterprise information systems*, 2018, publisher: Taylor & Francis.

[27] S. Besrour and I. Ghani, "Measuring Security in Requirements Engineering," *International Journal of Informatics and Communication Technology*, vol. 1, no. 2, pp. 72–81, 2012.

[28] D. R. Wallace, L. M. Ippolito, and B. B. Cuthill, *Reference information for the software verification and validation process*. DIANE, 1996.

[29] T. Diamantopoulos and A. L. Symeonidis, *Mining Software Engineering Data for Software Reuse*. Springer Nature, 2020.

[30] N. R. Mead and T. Stehney, "Security quality requirements engineering methodology," *ACM SIGSOFT Software Engineering Notes*, 2005.

[31] A. Björkelund, L. Hafdell, and P. Nugues, "Multilingual semantic role labeling," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, 2009.

[32] B. Bohnet, "Top accuracy and fast dependency parsing is not a contradiction," in *Proceedings of the 23rd international conference on computational linguistics (coling 2010)*, 2010, pp. 89–97.

[33] M. Luisa, F. Mariangela, and N. I. Pierluigi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering*, vol. 9, no. 1, pp. 40–56, 2004, publisher: Springer.

[34] GlobalPlatform Technology, "SESIP Security Evaluation Standard for IoT Platforms," GlobalPlatform Technology, Tech. Rep., 2021.

[35] D. Clegg and R. Barker, *Case method fast-track: a RAD approach*. Addison-Wesley Longman Publishing Co., Inc., 1994.