2023 IEEE 17th International Conference on Semantic Computing (ICSC) | 978-1-6654-8263-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSC56153.2023.00048

Validating Security Requirement Specifications through the use of a Knowledge Graph

Lynn Vonder Haar
Department of Electrical Engineering and Computer Science
Embry-Riddle Aeronautical University
Daytona Beach, U.S.A.
vonderhl@my.erau.edu

Sarah Reynolds
Department of Electrical Engineering and Computer Science
Embry-Riddle Aeronautical University
Daytona Beach, U.S.A.
reynos23@my.erau.edu

Tyler Procko
Department of Electrical Engineering and Computer Science
Embry-Riddle Aeronautical University
Daytona Beach, U.S.A.
prockot@my.erau.edu

Omar Ochoa
Department of Electrical Engineering and Computer Science
Embry-Riddle Aeronautical University
Daytona Beach, U.S.A.
ochoao@erau.edu

***Abstract*—Turning a customer's product vision into implementable requirement specifications is notoriously difficult and time-consuming. Beyond that, security is an aspect of requirement specifications that is frequently underdeveloped. If developers had access to an automated system that could validate requirement specifications based on known loopholes and external expertise, the likelihood of injecting a security vulnerability into a software product would decrease. As depicted in this paper, the automated system takes a Software Requirement Specification (SRS) document as input. The system then potentially validates that all related security topics and vulnerabilities are covered within the security requirement specifications. Entity linking is used to parse the SRS to form queries for a security concepts and vulnerabilities knowledge graph. The query result can provide developers with areas of security to be considered as updates to the SRS and to help during the software design stage to improve the system's security.**

Keywords—Entity Linking, Knowledge Graphs, Requirement Specifications, Requirements Validation, Software Security

I. INTRODUCTION

Converting a customer product vision into implementable requirement specifications poses a significant problem and time sink to software development. As a part of the requirements engineering process, requirements validation is essential in ensuring that the requirements are well-defined and meet quality standards [1]. In particular, validating security requirement specifications is a major resource sink due to software security's complexity and ever-changing nature [2]. To create accurate and thorough security requirement specifications, a development team needs expertise in software security as well as software development. While possible, the required level of expertise needed to create complete and quality security specifications can put many software teams at a disadvantage and lead to a low-quality product. An automated security specifications validation process can make a difference in a software project's time commitment and budget while giving an increased level of confidence in the system's security to those involved in a product. The work presented in this paper proposes a method to automate specification validation for security requirements using entity linking and a knowledge graph. This method discovers potentially absent security requirement specification topics by querying a Software Requirement Specification (SRS) document with terms representing keywords and concepts in relation to major security concerns and weaknesses. These terms are developed via entity linking, and the terms are used to query a knowledge graph based on a security concept ontology defined in this paper. Results from the knowledge graph match the requirement specifications to known security vulnerabilities. This work presents an approach through the specific example of user authentication, but the same approach could be used to expand to other weaknesses. The rest of this paper goes into more detail on the suggested method. Section two includes background information on requirement specifications, specification validation, entity linking, and knowledge graphs. Section three describes the method of this study. Section four describes the results we found when implementing this method. Section five discusses related work on this topic. Section six proposes future research into this topic, and section seven concludes the paper.

II. BACKGROUND

A. Requirement Specifications

Requirement specifications are detailed and technical statements describing the product's functional behavior and nonfunctional characteristics. These specifications contain the technical details that developers must implement to fulfill the requirements of the customer, which are often elicited from the customer's product description and the needs and desires of stakeholders. Writing requirement specifications can be time-consuming, challenging, and expensive. However, the presence of incorrect or missing specifications can lead to extensive bugs and potential defects in a system due to developers' misunderstanding of the requirements during their implementation [3]. It is of utmost importance to catch potential defects at the earliest stages of product development, as fixing defects becomes more expensive as the project develops [4]. The introduction of bugs into any system, but especially when discussing security, is detrimental, so writing robust and detailed security specifications is necessary for the system's

978-1-6654-8263-9/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSC56153.2023.00048

success. For this reason, though writing requirements specifications is costly, it is less costly than not writing them. However, it is always in the development team's interest to find methods and tools to aid the speed and accuracy of requirements specifications elicitation.

B. Security Requirements

Security is a nonfunctional requirement. The domain of the system and its priorities affect the level of security needed. High-risk systems, such as safety-critical or financial systems, require a higher level of security than other systems, such as standard websites.

Security, however, comes at the cost of other nonfunctional requirements. This cost is often usability or performance. For example, hashing and encryption are security measures that will affect the performance of a system because it needs to do extra calculations. Requiring dual verification for logging into an account is a security measure that affects the system's usability because it puts more work on the user to gain access to their account. Depending on the criticality of the domain, the usability and performance losses may be worth the extra security. Because of this, it is often hard to correctly integrate security features late in the lifecycle of a software system.

High-level nonfunctional security requirements are broken down into lower-level functional requirements that implement the nonfunctional security specifications.

The purpose of security specifications is often split into three categories: confidentiality, integrity, and availability, or the CIA triad. Confidentiality refers to not allowing unauthorized users to view information, integrity protects against unauthorized modification of data, and availability prevents denial of service attacks [5]. Careful design of system specifications will address each category while balancing security desires with other nonfunctional requirements such as performance.

Due to the expertise needed to write security requirements, achieving a good balance between security and other nonfunctional requirements can be challenging. Stakeholders often do not have the background to voice their security needs, and development teams tend to lack sufficient expertise in security to foresee the necessary requirements to build a secure system [6]. Therefore, development teams could benefit from a system that addresses key security oversights in the SRS.

C. Requirement Specification Validation

The validation stage of requirements engineering works to ensure that the specifications meet quality standards, including correctness and completeness. The most popular validation methods include reviews and inspections, and most established methods used throughout the industry involve manual validation of the requirements specifications [7]. The work presented in this paper suggests a validation method that can be more effective than manual methods in the case of security requirements specifications. The suggested method is automated, thus saving time in implementation. It relies on a knowledge graph, thus providing teams with up-to-date expertise that may not be present within the software development team.

It is important to note that, as discussed earlier, the level of security incorporated into a system depends on its domain and other nonfunctional requirements. It is unlikely that any proposed system can determine definitively if a given system's security requirements are complete. Therefore, the method proposed in this paper should be used as a complement to writing security requirements and can influence the design phase. However, it remains the software developer's responsibility to balance security needs with other nonfunctional requirements.

D. Knowledge Graphs

The domain knowledge of a particular application can be represented in a knowledge graph. Knowledge graphs are structures defining specific instances of concepts as described in a domain ontology. These domain concepts form nodes of the graph, and their relationships constitute the edges. This graph can then be queried using search terms to reveal the concepts and relationships within that domain. This node-edge paradigm is concretized into subject-predicate-object *triples*, where the subject is the starting node, the predicate is the relationship, and the object is the ending node [8]. Specifically for this work, the domain of the knowledge graph is publicly known security weaknesses. The weaknesses are related to each other by natural language keywords found in SRSs. These keywords describe the system's functional behavior that might require security mitigations based on the weaknesses documented in the knowledge graph [9] [10]. The Common Weakness Enumeration (CWE) is a publicly available list of security weaknesses that details known design flaws that often plague software systems [11]. The CWE gives pertinent information on the weakness, including attack patterns of the weakness and potential mitigations. The CWE forms the basis of the knowledge graph, but the key terms for the entity-linking algorithm must be compiled separately, and the attack patterns must be found externally. This approach in this paper uses the Common Attack Pattern Enumeration and Classification (CAPEC) to add attack pattern information to the knowledge graph [12]. CAPEC is a list of common patterns used by adversaries to attack systems. Awareness of these attack patterns can help guide security requirement development by planning defense around system areas vulnerable to those attacks. Knowing the relationships between security weaknesses, attack patterns, and system functionality can help to find oversights in the security plan described in the SRS [10]. By starting with keywords for functional behavior from the SRS, the knowledge graph can be queried to expand and provide a complete security plan since many security vulnerabilities are already represented explicitly [13].

In a philosophical context, Ontology is "the study of the kinds of entities in reality and of the relationships that these entities bear to one another" [14]. An ontology is an artifact of computer science created to be a controlled vocabulary representing the types of entities in a given domain. An ontology defines the structure of a knowledge graph within a

specific domain [15]. The research presented in this paper defines an ontology for the domain of security weaknesses [15]. The ontology describes general, upper-level domain concepts, such as "weakness category." Later, specific classes of weakness categories, such as "authentication errors," will be instantiated in the knowledge graph.

The knowledge graph can be queried using the SPARQL Protocol and RDF Query Language (SPARQL), the query language for resource description framework (RDF) knowledge graphs [16]. The query returns security weaknesses related to system functionality and potential mitigations that could be incorporated into the SRS to reduce or remove them. Thus, software developers can use shared experience from other software developers to find security loopholes and design robust systems around them.

E. Entity Linking

This approach described in this paper proposes using entity linking to find the search terms for querying the knowledge graph. Entity linking compares natural language text to a knowledge base of terms to understand the semantics of the text. Entity linking is generally done in three steps [17] [18]:

1. Recognition of the terms from the knowledge base in the text.
2. Disambiguation, which cleans up confusion from the first step. This step chooses a single link if there are multiple possible links from one spot in the text to different terms in the knowledge base.
3. Ranking the terms based on a chosen metric, such as confidence [17] [18].

This paper compiled the entity linking knowledge base using synonyms, snowballing from "authentication." This process was done manually to ensure a high level of control. The manual piece of this development process handled disambiguation and ranking, leaving only recognition of the terms to the system when searching the SRSs. This method is feasible when working in a small domain but can be costly in terms of time when trying to create links for a larger number of concepts. For that reason, while this approach uses a manual entity linking process, future work should experiment with automatic methods of entity linking.

There are other automatic ways to build the knowledge base for entity linking. For example, there are enterprise systems, such as DBpedia's, based on Wikipedia's knowledge base, that can identify a wide variety of terms [18]. The result of such an entity linker would be a list of URIs as links to the referenced terms [19]. Wikipedia allows the linker to have a much broader knowledge base, but this could lead to more frequent disambiguation because of multiple entities with the same name [17]. The approach presented in this paper has its own domain-specific knowledge base with the goal of higher accuracy of the entity linker due to relatively low disambiguation. Additionally, a smaller knowledge base with less disambiguation should increase the system's performance.

The purpose of using entity linking in this work is to narrow down the full text of the SRS to a few key search terms that can be used to query the knowledge graph. The knowledge base used here consists solely of terms related to user authentication. However, it could easily be expanded to other security topics via synonyms or any other method for knowledge base construction.

III. METHOD

A. Constructing the Knowledge Graph

The goal of this work aims to provide software developers with instances of known security weaknesses via a knowledge graph. To do this, a security weakness must first be defined. An ontology was created describing a security weakness in terms of its name, category, explanation, attack pattern, key terms, and possible mitigations. The relationships in the ontology are

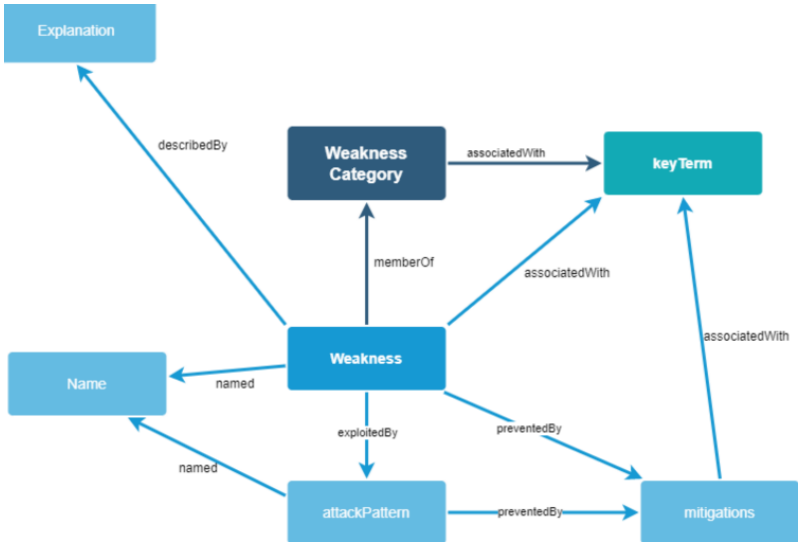

Figure 1. Weakness ontology.

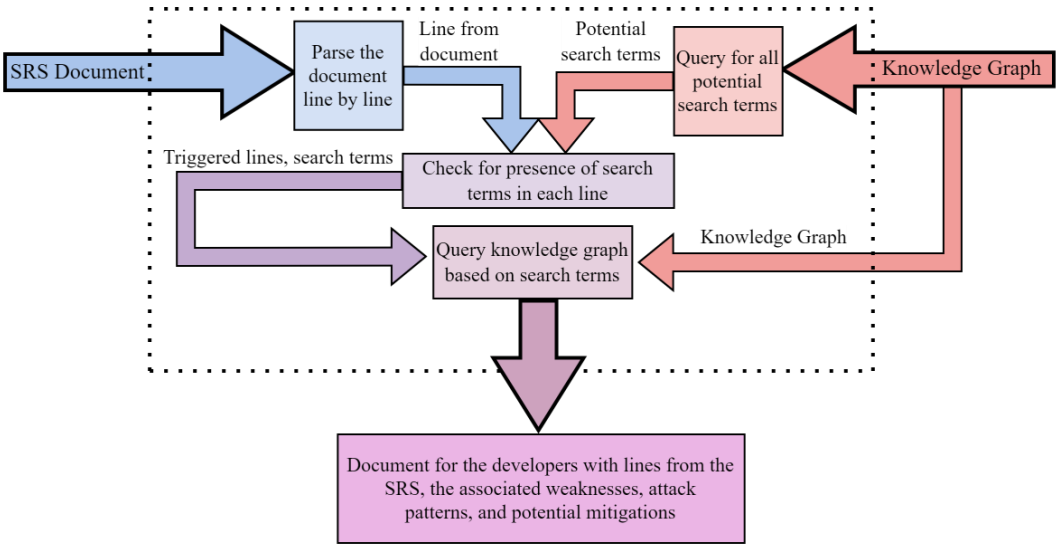

Figure 2. A depiction of the data flowing through the system.

uniquely defined for the purpose of this system. The ontology is shown in Figure 1. The ontology presented in Figure 1 was designed using information from the CWE. Pertinent information to avoiding the vulnerabilities was identified and the ontology was designed using that backbone of entities.

The knowledge graph then instantiates the ontology with specific weaknesses from the CWE. The knowledge graph documents the weakness based on the CWE identification number. Each weakness is linked to the overarching category "Authentication Errors" and is given the name and explanation provided by the CWE. When potential mitigations were provided within the CWE directory, these were also linked to the weakness. Otherwise, the mitigations node for that weakness was not included.

There is an established link between the weaknesses documented in the CWE and the attack patterns listed in the CAPEC. In the data provided within the CWE, attack patterns related to each weakness were provided via links. These links were adapted into our model, associating each weakness with related attack patterns. Those attack patterns were also linked to any potential mitigations provided in CAPEC documentation.

For this iteration of the process, the weakness categories were linked to key terms via manual entity linking. The terms added to the knowledge graph via this process are shown in Table 1. All of the terms in Table 1 are related to the act of user authentication since that is the focus of this work. While initially only using key terms as an entry point to weakness categories, the ontology provides the structure to link key terms to individual weaknesses and potential mitigations, which will prove useful in future iterations of this project. Potential future iterations are discussed more in the future work section.

It is important to note that the terms in Table 1 are all related to or synonymous to user authentication methods, such as logging in, username, and credentials. These are all broad topics that deal with what a system should do rather than how it should do it, which helps the system interface with and SRS document. Since an SRS document should also describe what a system should do, rather than how it should do it, these terms are likely to trigger the system when searching an SRS.

AuthenticationErrors cwo:AssociatedWith:				
accept user	access	access code	account	admit user
affirm user	approve user	authenticate	authenticity	authorization
authorize	authorized user	certified user	check user	confirm user
connect to	connecting to	control access	credentials	gain access
give access to	identity code	log in	log on	logging in
logging on	login	password	recognize user	sign in
sign on	signing in	signing on	user ID	user account
user admission	user confirmation	user credentials	user handle	user identification
user recognition	username	validate user	validation	verification
verify user				

Table 1. This table shows all of the key terms used that were associated with the weakness category "AuthenticationErrors."

The focus of this work is on the "authentication errors" weakness category, which includes 18 known security weaknesses from the CWE, the mitigations of those weaknesses, and associated attack patterns and mitigations provided in CAPEC documentation. The key terms for each weakness connect the knowledge graph to the natural language of the SRS. If the key terms are found in the SRS, they can be used to query the knowledge graph and find the related security weaknesses. The key terms were compiled manually using a snowballing effect of synonyms starting from authentication. These terms

form the knowledge base of the entity linking algorithm and become the search terms when querying the knowledge graph. This knowledge was applied to the ontology and was used to form a knowledge graph adhered to RDF. Therefore, the knowledge graph follows the organization of the ontology, but includes specific weaknesses found in the CWE.

The focus on authentication errors can further be expanded. The CWE provides other weaknesses and weakness categories that can be linked using our ontology to the existing knowledge graph. The categories, weaknesses, attack patterns, and mitigations are easily updated to represent the knowledge of an institution that adopts it. The key terms can be adjusted manually, or any of the described automatic entity linking methods can be used to develop key terms.

B. Requirement Specification and Knowledge Graph Integration

Once the knowledge graph has been established, a SPARQL query can be written to extract all potential key terms and phrases. The results of this query are used to search the SRS. Each requirement is searched, and if any of the key terms are found, both the key term and the requirement are saved as entry points to the knowledge graph. The knowledge graph is then queried to provide the developer with security weaknesses and mitigations related to the SRS.

It should be noted that in the tests of this process, the SRS documents are not limited to only providing requirements specifications, but other textual descriptions of the project, such as use case scenarios. Since SRS documents can be written in various formats, it was decided that the process needs to be effective in processing the whole document. Otherwise, the work of preparing the document to work with our system contrasts with the goal of this being a quick and easily implementable guide to improving security requirements. Therefore, the system processes the given document as a whole. Parts of the document that trigger a search are noted, and it is up to the user if they want to focus on a particular section of the document or not.

When a list of key terms present in the document has been compiled, they provide an entry point to the knowledge graph used by the SPARQL query. This query highlights weakness categories related to the project, specific weaknesses, and potential mitigations to prevent these weaknesses. Essentially, the developers are given a coverage report that shows which aspects of the system they are building could lead to security vulnerabilities. This document is organized by weakness category, then by weakness. Each requirement specification that triggered a relationship to a category or a weakness is grouped together and presented to the developer in coordination with the related weakness. This process is depicted in Figure 2.

As seen in Figure 2, the knowledge graph is queried twice. First, it is queried to find the list of all key terms related to all of the security weaknesses. Once those terms have been used to search the SRS, the knowledge graph is queried again using the relevant key terms to find the security weaknesses related to them. The resulting document includes not only the potential security weaknesses, their attack patterns, and their mitigations, but also the part of the SRS that triggered the weakness. This system provides the weaknesses to the developer but leaves it up to the developer to handle the weakness appropriately. Therefore, it would be up to the developer to consult their customer on which path to take. The system presented in this paper is meant only as a guide to highlight the areas of the system that introduce security weaknesses.

It is also important to note that this document only provides potential threats and gives light guidance. When it comes to security requirements specifications, they should only specify what the system does, not how the system will do it. The only exception to specifying the how in the requirements specifications is if it is something that the customer specifically requests. Therefore, this document may be used to highlight the vulnerabilities. When appropriate, it is up to the developer to decide how to amend the specifications. Potential mitigations may not be appropriate to include in the requirements specification but can be documented and referred to during the design phase. Additionally, this system returns all potential threats based on the key words found in the SRS. Therefore, it is up to the developer at this time to confirm whether they have addressed the potential threat in their requirement specifications, or if they should address it based on customer requirements. Automatic confirmation could be a direction of future work but is not covered by this approach.

This automated review process can provide an estimated percentage of completeness of security specifications. The system remains flexible by utilizing a knowledge graph to perform the analysis. As new security threats and mitigation methods appear, the knowledge graph can be easily updated to reflect the newest domain knowledge. As decisions are easily traced and sources of information are documented, the level of trust a user can have in this model is clear. The knowledge graph can be shared and updated between various entities, allowing each user set to decide the aspects and structure that work well for them.

IV. RESULTS

The system was tested on a selection of SRS documents written by students in a capstone-level class. Using the list of search terms provided by the knowledge graph, each document was searched for the presence of those terms, using the process outlined in Figure 2. If any terms were present within the document, the knowledge graph was queried to provide all related weaknesses and potential mitigations for the search term.

It is important to note that the key terms for this experiment were generated manually but independently from the tested SRS documents. It was not until after the key terms were generated that any of the SRS documents were tested and read. The SRS documents did not influence the generation of the key terms, which was essential to show a practical implementation of the system. Suppose a small set of documents provided the influence in key terms and the testing results. In that case, the system could not be depended on to provide equivalent results to external applications.

The SRS documents testing set comprised 62 SRS documents created by undergraduate students. These

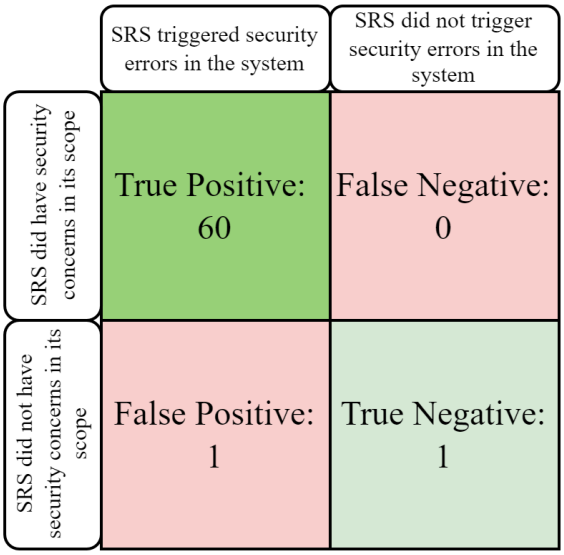

Figure 3. A depiction of the results of running 62 student-written SRS documents through the knowledge graph system.

documents included requirement specifications and supporting diagrams such as use case scenarios. The entirety of the document was tested on our system to determine if any of the projects described had requirements related to authentication of users, which would trigger a category of security weaknesses related to “Authentication Errors” and provide a list of potential weaknesses and possible mitigations for the developer.

The document returned to the developer provided the results to the reader organized by weakness. All weaknesses were sorted by category, and potential mitigations were listed directly following the weakness. Additionally, any requirements that triggered the weakness or weakness category were listed, with the triggering key words highlighted in bold. This format, which is shown in Figure 4, was chosen to allow the developers to clearly see the results, while maintaining traceability between the weaknesses and the original SRS document.

The 62 documents were initially tested on a graph with a singular key term entry point to "Authentication Errors.” This one access point triggered 21 of the 62 SRS documents to have potential security concerns related to authentication. When the system was tested with the complete list of 47 access terms, 61 out of the 62 documents were triggered with potential security concerns. The system accuracy of identifying security weaknesses from the SRSs is summarized in Figure 3.

Out of the 61 documents, multiple statements within each document triggered the "Authentication Errors" category. Only one of the documents was incorrectly triggered because the key terms that were triggered were specifying that no authentication was needed to access the system.

SRS Security Concerns Report

1. Weakness Category #1
    a. Weakness A
    b. Weakness B
        i. Potential Mitigation
        ii. Potential Mitigation
    c. Related Specifications
        i. Specification
        ii. Specification
2. Weakness Category #2
    a. Weakness C
    b. Weakness D
        i. Potential Mitigation
    c. Related Specifications
        i. Specification
        ii. Specification

Figure 4. A depiction of the format of results that are returned to developers.

Additionally, in the one document that was not triggered to have security concerns, there was nothing in the entire document that discussed the authentication of users. While the requirement to authenticate users at some point may have been an overlooked specification, that is outside this system's scope. The system was correct not to identify that there may be security concerns related to authentication.

The results show a successful implementation of the proposed approach. It correctly identified the correct scenarios to bring up potential security concerns. In the cases used, security concerns were largely ignored in the SRS document. This behavior can be expected of undergraduate students. However, it also provides an example of how the system can be used to supplement developers who do not have a sufficient level of security expertise. This paper earlier addressed the problem that software projects will have security concerns, while not all software development teams have the security expertise necessary to appropriately address these security concerns. By testing the SRS documents of a group of novice software engineers, these results suggest that the process outlined in this paper can correctly identify potential security concerns for a software project.

V. Related Work

Some papers cover the importance of requirement verification as a quality assurance method [20] [21]. One such paper discusses using requirement decomposition to form an ontology from the requirements [20]. Decomposition helps to break the requirements down to be more atomic. This ontology of concepts contained in the requirements can then be used to verify that the system meets each of the requirements [20]. However, this is different from the method and purpose of this paper, which uses a knowledge graph containing related security concepts, accepts an SRS as input, and validates the completeness of the requirement specifications in the SRS. This system is used within the design phase rather than after the

product has been developed. Another paper on verification describes a method to create an ontology based on the requirement specification structure for system verification [21]. In contrast, this paper uses semantics to form the knowledge graphs and is used in specification validation.

Another paper discusses the recent rise in the importance of information security and how shared knowledge can be used to track known security loopholes in software development [22]. The mentioned paper suggests an approach to sharing knowledge using the semantic web because it offers a universal language and structure for sharing knowledge [22]. The goal of the project is to combine information security repositories with software build repositories so that software projects automatically know about open-source security loopholes [22]. In contrast, this paper uses the semantic web to suggest missing security requirement specification topics by parsing an SRS.

In another paper, they created their own software ontology called the Software Evolution Ontology and combined it with the semantic web to help in software analysis [23]. Specifically, this ontology works to track the evolution of a software project throughout its development [23]. This ontology is then queried using a specialized querying language, iSPARQL [23]. This paper differs by ontology domain and application. This paper focuses on concepts related to security development to write requirement specifications versus evolution analysis.

There are other examples of security vulnerability ontologies available including the SEPSES knowledge graph and the security ontology. The SEPSES knowledge graph uses some of the same resources that the ontology built here uses, such as CWE and CAPEC [10]. However, it also uses several other knowledge bases. The goal of the SEPSES knowledge graph is to make a broader knowledge base for cybersecurity concepts that creates links between previously unlinked datasets. Their knowledge base, though centered around the same central resources that this work uses, lacks the entity linking concepts. In the future, it would be possible to integrate the two systems, allowing the system proposed in this paper to work with a larger knowledge base. The security ontology also models a security threat, its source, its potential severity, and its location [24]. The security ontology also includes entities describing companies, their data, and their employees. Much like the SEPSES knowledge graph, the security ontology is larger than the ontology suggested in this paper and could overwhelm the user. Additionally, the security ontology is standalone without the automatic SRS checker.

VI. Future Work

Expanding the knowledge graph of security weaknesses and the knowledge base of the entity linking algorithm is the next step in this process. This system could then be utilized for a more comprehensive review of SRSs. This expansion can be accomplished by using CAPEC, CWE, and other community security repositories to expand the coverage of the knowledge graph. If security experts take this structure and build their own knowledge bases, that information can be incorporated within the knowledge base- provided the user determines that it is coming from a trusted source. As security vulnerabilities are mitigated, the community repositories are updated, enabling the currency of this project's knowledge graph.

Alignment of the proposed knowledge graph of security weaknesses to a well-established, quality upper-level ontology, such as the Basic Formal Ontology (BFO), or its expanded project, the Common Core Ontologies (CCO), would support integration with future projects in disparate domains, and expansion to related concerns plaguing SRS development, e.g., reliability. This step would accompany the expansion of the knowledge graph described above. Seeking future semantic interoperability, the use of relationships from an ontology such as the Provenance Ontology (PROV-O) should be sought out.

A method to test the accuracy and reliability of the system is needed. The system could suggest any security requirement specifications, but not necessarily the right ones to cover the system's needs. While an expert in the field could check the results of the query, their review could be biased and might not consider all known security vulnerabilities. For now, the system's performance is limited by the trust a user has in the knowledge graph itself, as well as the ability of the user to interpret the results that this system provides.

Developing a standard automated test is another area for future research. This testing method aligns with the tradeoff between security and other nonfunctional requirements. Although this process may suggest security mitigations to reduce or remove security weaknesses from the system, those mitigations may introduce unacceptable consequences to the usability or performance of the system. Balancing the security rewards and consequences is still up to the software developer, so expanding this process to understand the domain of the software system and making more relevant suggestions could be a future research direction.

SRS documents written by undergraduate students provided excellent material for testing this system. It is rare, if not impossible, to get individuals with enough knowledge and experience within a group of undergraduate students to write effective security requirements. The current state of the experiment highlighted many potential security threats developed by students. After expanding the system to handle more categories and instances of security weaknesses, this system could be provided to students while they are engaged in the class. Then, data could be collected on how the security requirements change after implementation of this system, as well as survey results to document the student's response to and feelings towards the effectiveness of this system.

This system can also be expanded to not only trigger potential security concerns but to check the provided document to see if the security concerns have been handled. Once the document has been processed and the user has a list of potential threats and potential mitigations, the document can be searched once again to see if the potential mitigations are mentioned within the requirements specifications. To do this successfully, the knowledge graph would have to be developed to link key terms and phrases to each potential mitigation. This process would be very similar to the one described in this paper. As a result, the developer would still get a list of potential security issues. If those are already addressed within the document, that

information will be noted along with where they are addressed. To test this proposed change to the system, it will need to be tested on a wider variety of SRS documents, including documents written by individuals with a higher level of security expertise.

VII. Conclusion

Automating the validation of requirement specifications could save a significant amount of time and money in industry. In the case of security specifications, it could also prevent the injection of publicly known security vulnerabilities into the system. The process proposed in this paper takes an SRS as input and uses key security terms found in the SRS to query the knowledge graph. Through this process, the security vulnerabilities and potential mitigations present within an SRS document are provided to the developer. The result of this query highlights security threats that are relevant to the system and estimates the completeness of security specifications as a method of validating the SRS. This paper included a proof of concept that demonstrated the effectiveness of the process. Future work includes expanding the knowledge graph and additional testing.

References

[1] S. Maalem and N. Zarour, "Challenge of validation in requirements engineering," *Journal of Innovation in Digital Ecosystems,* no. 3, pp. 15-21, 2016.

[2] D. Pandey, U. Suman and A. K. Ramani, "Security Requirement Engineering Issues in Risk Management," *International Journal of Computer Applications,* vol. 17, no. 5, 2011.

[3] W. M. Wilson, L. H. Rosenberg and L. E. Hyatt, "Automated Analysis of Requirement Specifications," in *International Conference on Software Engineering*, Boston, 1997.

[4] S. Lipner, "The Trustworthy Computer Security Development Lifecycle," in *20th Annual Computer Security Applications Conference*, Tucson, 2004.

[5] S. Samonas and D. Coss, "The CIA Strikes Back: Redefining Confidentiality, Integrity, and Availability in Security," *Journal of Information System Security,* vol. 10, no. 3, pp. 21-45, 2014.

[6] S. H. Houmb, S. Islam, E. Knauss, J. Jurjens and K. Schneider, "Eliciting Security Requirements and Tracing Them to Design: An Integration of Common Criteria, Heuristics, and UMLsec," *Requirements Engineering,* vol. 15, pp. 63-93, 2010.

[7] H. Bilal, M. Ilyas, Q. Tariq and M. Hummayan, "Requirements Validation Techniques: An Empirical Study," *International Journal of Computer Applications,* vol. 148, no. 14, pp. 5-10, 2016.

[8] "A Review of Relational Machine Learning for Knowledge Graphs".

[9] J. Bailey, F. Bry, T. Furche and S. Schaffert, "Web and Semantic Web Query Languages: A Survey," *Reasoning Web,* pp. 35-133, 2005.

[10] E. Kiesling, A. Ekelhart, K. Kurniawan and F. Ekaputra, "The SEPSES Knowledge Graph: An Integrated Resource for Cybersecurity," in *International Semantic Web Conference*, Auckland, 2019.

[11] "Common Weakness Enumeration (CWE): A Community-Developed List of Software and Hardware Weakness Types," MITRE, 2022. [Online]. Available: https://cwe.mitre.org/. [Accessed 3 October 2022].

[12] "Common Attack Pattern Enumeration and Classification," MITRE, 2022. [Online]. Available: https://capec.mitre.org/. [Accessed 20 October 2022].

[13] M. Cadariu, E. Bouwers, J. Visser and A. van Deursen, "Tracking Known Security Vulnerabilities in Proprietary Software Systems," in *International Conference on Software Analysis, Evolution, and Reengineering*, 2015.

[14] R. Arp, B. Smith and A. D. Spear, Building Ontologies with Basic Formal Ontology, MIT Press, 2015.

[15] X. L. Hainan Chen, "An automatic literature knowledge graph and reasoning network modeling framework based on ontology and natural language processing," *Advanced Engineering Informatics,* vol. Volume 42, 2019.

[16] "SPARQL 1.1 Overview," W3C, 21 March 2013. [Online]. Available: https://www.w3.org/TR/sparql11-overview/. [Accessed 27 October 2022].

[17] D. Ceccarelli, C. Lucchese, R. Perego, S. Orlando and S. Trani, "Dexter: An Open Source Frameword for Entity Linking," in *Sixth International Workshop on Exploiting Semantic Annotations in Information Retrieval*, 2013.

[18] J.-C. Klie, R. Eckart de Castilho and I. Gurevych, "From Zero to Hero: Human-In-The-Loop Entity Linking in Low Resource Domains," in *58th Annual Meeting of the Association for computational Linguistics*, 2020.

[19] G. Munnelly and S. Lawless, "Investigating Entity Linking in Early English Legal Documents," in *18th ACM/IEEE on Joint Conference on Digital Libraries*, 2018.

[20] H. Hu, L. Zhang and C. Ye, "Semantic-based Requirements Analysis and Verification," in *International Conference on Electronics and Information Engineering*, 2010.

[21] R. Chen, C.-H. Chen, Y. Liu and X. Ye, "Ontology-based Requirement Verification for Complex Systems," *Advanced Engineering Informatics,* vol. 46, 2020.

[22] S. S. Alqahtani, E. E. Eghan and J. Rilling, "Tracing Known Security Vulnerabilities in Software Repositories - A Semantic Web Enabled Modeling Approach," *Science of Computer Programming,* vol. 121, pp. 153-175, 2016.

[23] C. Kiefer, A. Bernstein and J. Tappolet, "Mining Software Repositories with iSPARQL and a Software Evolution Ontology," *IEEE,* 2007.

[24] S. Fenz and A. Ekelhart, "Formalizing Information Security Knowledge," in *4th International Symposium on Information, Computer, and Communications Security*, 2009.