Identifying Security Concerns Based on a Use Case Ontology Framework

Imano Williams, Xiaohong Yuan
Computer Science
North Carolina Agricultural and Technical State University
Greensboro, U.S.A
irwilli1@aggies.ncat.edu, xhyuan@ncat.edu

***Abstract*— Identifying security concerns in an application can be difficult, especially if the analysts lack security knowledge. We propose a use case ontology that can help to identify security concerns based on the use case specifications. We demonstrate the feasibility of the ontology by systematically applying the ontology to use case specifications expressed in Web Ontology Language (OWL). The proposed approach can help model the interrelationship of concepts in the use case and possibly use queries to group use cases that may have similar security concerns. This approach could allow analysts to identify parts of the use cases with similar security concerns and could potentially reduce reoccurrences of known vulnerabilities in software applications. Lastly, we discuss future work about creating an automated tool for recommending attack patterns for the security requirements process.**

Keywords-Domain Ontology; Security Concerns; Use Cases; Secure Software Engineering

I. INTRODUCTION

Ontological modeling of software artifacts has been used in the requirements and design phases to address security issues [1]. Some ontologies have used software artifacts such as security requirements [2] and use cases [3] to aid in knowledge acquisition and the conceptualization of reusable domain-specific software security information. According to Veres, et al. [4], ontologies can be used to track the dependencies between requirements as the project becomes realistically complex. However, when ontologies are used to elicit security requirements, the security requirements elicited depend on the ontology that was used [5]. Furthermore, requirement analysts may not have adequate security knowledge to choose the most appropriate ontology and then use it correctly [6]. In addition to security knowledge, requirement analysts also need domain knowledge to identify security concerns. Therefore, the modeling of the system based on an ontology can be difficult for security requirement analysts.

In this paper, we introduce a use case ontology framework to identify security concerns based on use case descriptions. The following observations motivated us to build the ontology: (1) Reports from Open Web Application Security Project (OWASP) Top 10 [7] and Common Vulnerability Exposure[1] (CVE) have shown the frequent reoccurrences of known vulnerabilities, such as SQL Injection and Cross-site Scripting. (2) Many of the reported vulnerability exploits started from the web interface of an application. (3) In a software development team, different understandings of what to secure in software under development (SUD) may lead to ambiguous, incomplete, and inconsistent security concerns being identified by the stakeholders.

We created the ontology framework for identifying security concerns by (1) Identifying the Assets and Web Components of a use case that can guide requirements analysts to raise security concerns via use case steps; (2) Creating concepts and attributes for the proposed ontology based on the results of steps (1); and (3) Associate the use case steps (or flows) to specific security concerns using semantic rules; (4) Based on this ontology, semantic queries can be run to find similar use case flows that may have similar concerns. This ontology framework can help reduce reoccurrences of know vulnerabilities by identifying similar security concerns across different functionalities of an application and different applications.

The rest of the paper is organized as follows: Section II presents the proposed ontology. Section III defines rules for identifying security concerns. Section IV discusses how the ontology can be used to identify security concerns. Section V demonstrates how the ontology framework is used with a specific example. Section VI discusses related work. Finally, Section VII concludes the paper and discusses future work.

II. THE PROPOSED USE CASE ONTOLOGY

The proposed ontology was developed using the steps suggested by Noy and McGuinness [8] with an evolutionary approach. These steps include defining the scope of the ontology, reusing existing ontologies, enumerating important terms, defining classes, defining properties, defining cardinality, and creating instances (individuals). We adopted concepts that are related to security concerns in [9], the Restricted Use Case Model (RUCM) [10], and verb categories for web tasks from [11-13] to develop the ontology. The RUCM is a use case template that specifies 26 restriction rules on the natural language, keywords for control structures, and that every flow path in a use case should have a postcondition. In addition to using a more restrictive use case template, we used a dialog descriptive use case format. The dialog use case format includes a graphical user interface (UI) components in the use case flow.

[1] https://cve.mitre.org

DOI reference number: 10.18293/SEKE2020-136

The rationale for using dialog descriptive use case includes the following: (1) the web components in the use case could guide requirements analysts to raise security concerns; (2) use cases are rudimentary software development artifacts that can represent the system navigational structure from a graphical user interface aspect; (3) according to Salini and Kanmani [14], the user interface and navigational structure are the main features of applications' web interfaces that must be analyzed and (4) currently, we focus on the constrained system where the design of the interaction is more precise than just providing the intent of the use case.

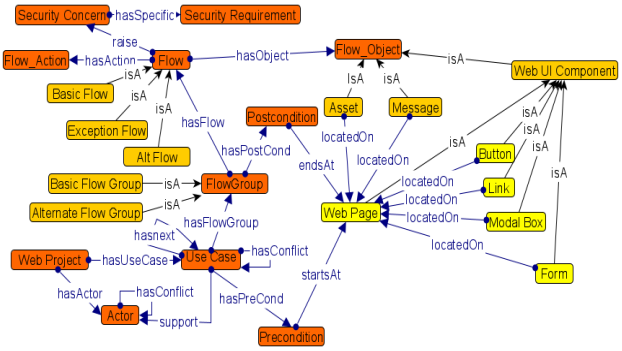

Figure 1 The proposed use case ontology for identifying security concern

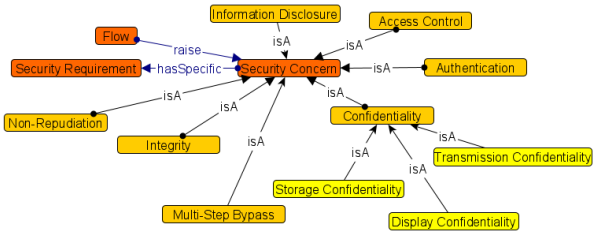

Figure 2 Security concern subclasses

In Fig. 1, we have shown some of the major concepts that were taken from different sources to build the proposed ontology. Fig. 2 shows the concepts related to security concerns. These security concern concepts were adopted from ISO/IEC 27001:2013 [15], ISO/IEC 27000:2018[2], and [9]. Next, we provide the definitions of some of the core concepts:

- **Use Case**: Represents the intended interactive steps between an external entity and the system.
- **Actor**: Represents a human or an external system that interacts with the system to accomplish the services of the use case.
- **Flow**: It specifies the logical steps that an actor takes to complete the services of the use case.
- **Web UI Component**: An interface component of the application that actors interact with to complete the services of the use case. In our ontology, we created concepts for the button, link, web page, and modal box.
- **Flow Action**: This is an operation that is performed by the Actor or the SUD to complete a Flow. Some examples are "*The user updates the username.*" and "*The system displays the 'login'* web page."
- **Asset/Message**: An intangible valuable resource, such as a password that is worth protecting. Here we focus on data the user provides via some user input.
- **Security Concern**: Matters of interest related to security exploits that may affect use case flows based on the action, web components, and the asset. The subclasses are authentication, authorization, confidentiality, integrity, non-repudiation, identity, and security auditing.
- **Security Requirements**: Conditions that must be satisfied to address a security concern.

An object property is represented as "o (D → R)", which means a class D (domain) is related to another class R (range) by o, the object property. Some of the object properties are:

- **hasFlow** (FlowGroup → Flow)
- **hasFlowBefore** (Flow → Flow), an inverse of **hasFlowAfter** (Flow → Flow)
- **hasActor** (Use Case → Actor)
- **raise** (Flow, Use Case, Post Condition → Security Concern)
- **display** (System → Web Page, Modal Box, Message)
- **validate** (System → Asset)
- **hasFlowObject** (Flow → Web UI Component, Message, Asset)

A data property is represented as, "d (C → r)", which means that class C has data property, d, with range r. Some of the data properties are:

- **hasActionType** (Action → ["passive", "active"])
- **hasLinkParameter** (Link → ["non-sensitive", "sensitive", "both", "none"])
- **hasInformation** (Asset, Web Page → ["non-sensitive", "sensitive", "both", "none"])
- **hasAppLocation** (Use Case → ["authenticated", "unauthenticated", "both"])
- **hasPurpose** (Use Case → ["create", "read", "update", "delete"])
- **hasInteractionFlow** (Use Case → ["multi", "single"])
- **isValidationFlow** (Flow, → ["yes", "no"])

To formally specify the classes (concepts), along with their object properties, data properties, and quantifier restriction, we used the Web Ontology Language (OWL) [16]. We used the second level OWL 2, OWL-DL, which provides maximum expressiveness while retaining the inference capabilities of an ontology [17] and semantic queries over the knowledge.

III. Rules for Identifying Security Concerns

We defined rules using Semantic Web Rule Language (SWRL) [18] to identify security concerns based on the assets,

[2] https://standards.iso.org/ittf/PubliclyAvailableStandards/

web UI components, and the actions in the use case flow. Currently, we defined eight rules for identifying security concerns that are listed below. These rules are not exhaustive.

Rules for the use case flow concepts:

1) *Asset <sensitive>* && Save → *Storage Confidentiality*, which means that an Asset instance with a "sensitive" data property value that should be saved has storage confidentiality.
2) *Asset <sensitive>* && *Display* → *Display Confidentiality*, which means that an Asset instance with a "sensitive" data property value should have display confidentiality.
3) *Asset* && *Validate* → *Multi-Step Bypass*, which means that an Asset instance that is being validated by the system multi-step bypass. This security concern occurs when a user can bypass some validation logic to get to another flow in the use case.
4) *Button <active>* && *Click* → Non-*Repudiation* && *Transmission Confidentiality*, which means that a Button instance with an active data property (i.e., it makes changes to the system file system) associated with a click action have non-repudiation and transmission confidentiality.
5) *Link <sensitive>* && *Click* → *Transmission Confidentiality*, which means that a click action on a Link instance with "sensitive" parameter data property value has transmission confidentiality.
6) *Button*, *Link* && Click → *Information Disclosure*, which means when the system generates messages (warning, error, or confirm) because of an action on a link or button has Information Disclosure.

Rules for the use case concept:

7) hasAppLocation *<authenticated* or *both>* → Authentication, which means a Use Case instance with authenticated or both boundary type has of authentication.

Rules for the actor concept:

8) *Conflicting Use Cases* && *Actor* → *Separation of Duties*, which means a use case that is followed by a conflicting use case, should not have the same Actor. Conflicting meaning that the same Actor cannot use two or more use cases. This rule is an extension of Rule 2.

Apart from having these eight rules, we can create informal rules (not in SWRL) to be used as SPARQL Protocol, and RDF Query Language (SPARQL) query, such as:

1) *Non-Permitted Actor* && *Web Pages* → *Inaccessible Web Pages* means that a user role is not permitted to view restricted web pages.

IV. The Proposed Ontology Framework Approach

In this section, we describe how the ontology could be in a framework to identify security concerns. Fig. 3 shows the three major phases, along with the respective sub-phases.

In the first phase, *Identifying Instances*, the instances based on the use case concepts are identified along with their object and data properties. We start by identifying the name of the use case along with its data properties, and then we find the instances that are related to the use case via its object properties.

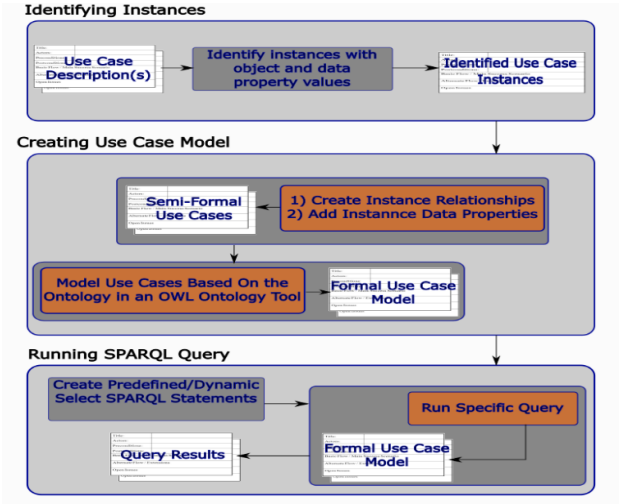

Figure 3 The Proposed Ontology Framework

For example, we identify the Actor instance and then find its data and object properties. Overall, we perform a depth-first identification of the instances (via concepts) with the data and object properties, then recursively perform a depth-first identification on the next instance that is related to the current instance via the object property. As a result, we identify the Asset, Action, and Web UI Component from the flows of the use cases along with their inter-relationships. For example, we can specify the *hasFlowBefore* and *hasFlowBefore* object properties for the current flow or the web pages that an asset is located.

In the second phase, Creating Use Case Model, the output of the first phase is used to create the semi-formal Resource Description Framework (RDF) triples (*subject, predicate, object*) of use case descriptions. Next, Protégé [19], the OWL 2 editor, is used to create the instances and their object and data properties that were identified in the first phase. During the second phase, the Pellet[3] reasoner is run regularly to continually check the consistency of the asserted facts being added in the [20] ABox (asserted facts about the use cases) of the ontology.

In the third phase, Running SPARQL Query, SPARQL is used to query the ABox to find security concerns based on the inferred facts using the SWRL rules in Section III. For example, if several Flow instances can be affected by Multi-Step Bypass, the query will return those Flow instances. We used the approach proposed by Uschold and Gruninger [21] to evaluate the ontology based on motivating scenarios, informal competency queries, and formal competence queries to help identify security concerns for the modeled application. To run SPARQL queries, we use Snap-SPARQL [22] plugin[4] in Protégé that supports reasoner inferences using the Pellet plugin. Pellet, an OWL reasoner plugin in Protégé, is used to assist in answering the queries about the security concerns.

[3] https://protegewiki.stanford.edu/wiki/ProtegeReasonerPlugin

[4] https://github.com/protegeproject/snap-sparql-query

V. Demonstrating the Ontology Framework

A. A Case Study

We used a mock online [5] shopping web application to demonstrate the ontology framework. The total number of instances (individuals) modeled were 11 use cases, four actors, 22 web pages, 24 assets, 17 links, eight buttons, 90 flows, and 14 actions in the proposed ontology. We had to edit the use cases to conform to the standards of the Restricted Use Case Model [10]. The edited use cases and ontology files are located at Use_Case_Ontology_for_Security_Concern[6]. Fig. 4 shows a description of the "*Create New Account"* use case. In the use case description, we used a dashed line to mark the action and single underline to mark the object in the sentence to be modeled in the ontology as triples.

USE CASE UC3: Create New Account
Actors: Primary – Unregistered Customer
Preconditions: The system is displaying a 'Home' Webpage to the user.
Basic flow:
1. The user clicks on 'Create New Account' link.
2. The system displays the 'New Account Information' screen.
3.The user enters the FirstName, Last Name, Street Address, City, State, Country, Postal Code, Card Number, Card Type, and Card Expiry Date.
4. The user clicks on the 'Create' button.
5. The system validates that the FirstName, Last Name, Street Address, City, State, Country, Postal Code, Card Number, Card Type, and Card Expiry Date are correct.
6. The system displays the 'Signup Information' webpage.
7. The user enters the username and password.
8. The user clicks on the "Sign Up" Button.
9. The system validates that the username does not exist.
10. The system displays the 'Account Confirmation' webpage along with the FirstName, Last Name, Street Address, City, State or Province, Country, Postal Code, Telephone Number, Card Number, Card Type, and Card Expiry Date.
11. The user clicks on the "Verification" button.
12. The system displays the 'Account Information' Page.
Post Condition: The system saves the FirstName, Last Name, Street Address, City, State or Province, Country, Postal Code, Telephone Number, Card Number, Card Type, and Card Expiry Date. The system is displaying the 'Account Information' Page to the user.
Bounded Alternate Flow: N/A
Global Flow: N/A
Specific Flow: N/A

Figure 4 The Create New Account Use Case

In the *Identifying Concepts* phase, we identify the concepts in the sequence of Use Cases → Actors → Preconditions → Flow Groups → Flow (subject, predicate, object) → Postconditions along with their data and object properties. The *has Boundary* data property of the "*Create New Account*" use case is on the "*unauthenticated*" side of the application. There is only one Actor, Unregistered Customer, for this use case. To initiate the use cases, the Actor would start from the "Home" page. Next, we move onto the basic flow to identify the predicate and object. The user is sending account information that the system must validate in basic flow four before the user can supply the signup username and password to complete the use case (basic flow 5). Therefore, the interaction is a "*multiple-step*" (temporal expression). We determine the Asset instance based on the data the actor supplied to the system or vice versa. The use case's purpose is "*active (inserting)*" since the use case is making changes to the file system to create the new account. Fig. 5 shows the representation of basic flow six, along with its object and data properties in the Protégé editor. The object property *hasFlowBefore* has value basic flow five. Additionally, the inferred flows before and after are shown through the *dependsOnFlowBefore* and *subsequentFlow* object properties.

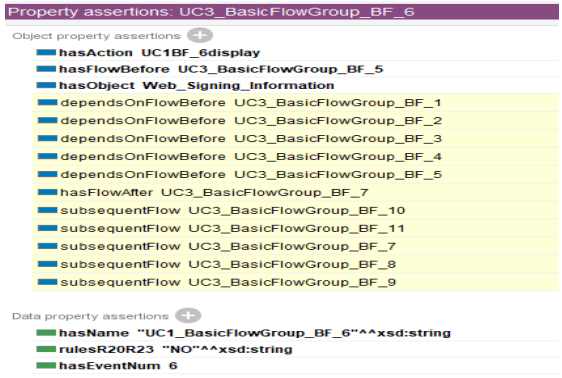

Figure 5 Basic Flow in Protégé

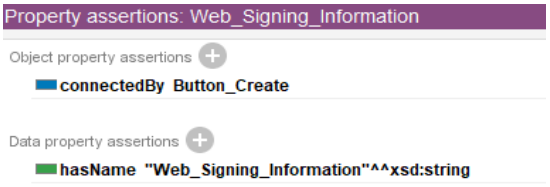

Figure 6 Signup Information Web Page in Protégé

Fig. 6 shows the Signup Information Web Page from basic flow six. The object property *connectedBy* shows the navigational path to get to the web pages. In this case, it is the "Create" button located in basic flow four. It could also be a Link instance. Furthermore, the "Sign Up" button in flow eight in Fig. 4 would have a *buttonLocatedOn* object property of Signup Information Web Page in the ontology.

Once we have completed the Creating Use Case Model phase for all the use cases, the next phase is Running SPARQL Queries based on the defined SWRL rules. We used SNAPSPAQRL to run the queries since it supports inferring once a reasoner is running. The SWRL rule for rule 3, Multi-Step Bypass is *Flow (?f), Asset(?a), Action (Validate), has Asset(?f, ?a), has Action(?f,Validate), Mult_Step_ByPass(?m)* → *raise (?f, ?m)*. We can then query ontology to find the consequent of the SWRL rule once the antecedent is true. So, we can have informal and formal queries to search:

1. **Informal Query**: Which flows are affected by the *Multi-Step Bypass security concern?*
2. **Formal Query** SELECT **?useCase ?flow** WHERE {
   ?useCase uc:hasFlowGroup ?flowGroup.
   ?flow uc:isPartOfGroup ?flowGroup.
   ?flow uc:raise uc:MultiStepBypass. }

TABLE I shows the partial results of running the above formal query.

TABLE I MULTI-STEP BYPASS QUERY RESULTS

Use Case	Flow Affected
UC11_Login	UC11_MF5
UC3_Create_New_Account	UC3MF5
UC3_Create_New_Account	UC3MF9

[5] https://personal.utdallas.edu/~chung/RE/Presentations07S/Team_3/

[6] https://figshare.com/projects/Use_Case_Ontology_for_Security_Concern/80330

The rules in Section III are not the complete SWRL ruleset. Apart from running the queries based on the defined SWRL rules, an analyst can also create and run new queries based on reported CVEs. For example, CVE-2018-14398[7], *an issue was discovered in Creme CRM 1.6.12. The value of the cancel button uses the content of the HTTP Referrer header and could be used to trick a user into visiting a fake login page to steal credentials*. This vulnerability is related to a user clicking a button component on the web page. Therefore, we can create a SPARQL query to find the pages that involve users clicking a button. The query would be the following:

1. **Informal Query:** Which flows display a button, and which pages are these buttons located?

2. **SPARQL Query:** SELECT **?flow ?button ?webpage** WHERE {**?useCase** uc:hasFlowGroup ?flowGroup.
?flow uc:isPartOfGroup ?flowGroup.
?flow uc:hasFlowObject **?button** .
?button a uc:Button ;
uc:buttonLocatedOn **?webpage** . }

From the query results in TABLE II, UC2_Search Catalog – basic flow three, UC9_Make Online Payment – basic flow (success path) six, UC3_Create New Account – basic flow four, UC4_Update Account Information – basic flow five, and UC8_Apply for Financing – basic flow six are affected by Rule 6 (information disclosure) when a user clicks a button. We can run similar queries to find other parts of the system that could be exposed to other CVEs.

TABLE II BUTTON IN USE CASE MODEL SIMILAR TO CVE-2018-14398

Flow	Button	Web Page
UC9_BF6	Button_Submit	Web_Make__Payments_Page
UC2_BF3	Button_Search	Web_Main_Page
UC3_BF4	Button_Update	Web_New_Acct_Information_Page
UC4_BF5	Button_Finish	Web_Update_Acct_Information_Page
UC8_BF6	Button_Submit	Web_Make__Payments_Page

B. *Discussion*

In section V.A, we demonstrate how the ontology framework can be applied to a specific use case. The object and data properties show that specific rules could be used to find similar parts of different use cases that may share the same security concern. So, we focus more on providing a modeling process to find common security concerns through specific scenarios inspired by reports from OWASP and CVE, where the interface of the application is concerned.

In terms of performance, the instance Identification and Use Case Model Creation phases of the framework are task intensive. We had to take precautionary steps so that we did not miss sub-tasks, such as extracting the information from the use case description that is related to the concepts in the ontology. Manually populating the ontology by copying the semi-model from the document to the ontology via Protégé is tedious and time-consuming. Also, it is easy for the ontology to become inconsistent when using the wrong individual for the range of an object property. Furthermore, an analyst may forget to add information from the semi-model. It took more than 2 hours to populate the ABox of the ontology for the 11 use cases. As a result, an automated process to identify what to populate the ABox of the ontology is needed to make the process take less effort by a user.

VI. Related Work

Gärtner, et al. [3] developed an integrative security knowledge model that identifies vulnerabilities from software requirements (use cases) based on reported security incidents. They conducted a case study that showed how different use cases were related to a similar misuse case, but their proposed knowledge structure was not able to identify interrelationship between use cases that may have the same misuse case. Rago, et al. [23] used text mining to identify quality attributes such as modifiability, performance, availability, security from use case description. Their work aimed to help requirements engineers skim through requirements documentation efficiently, in order to identify potential quality attributes such as performance, security, mobility, and testability. However, in terms of security quality attributes, they did not delve into security concerns. Wouters, et al. [24] proposed a semi-formal ontology for the reuse of similar use cases by defining labels, concepts and relations to create rules and queries in an inference machine to find similar use cases. Our work is similar to theirs in the conceptual model of user interaction with UI in use case. However, our ontology includes more detailed UI components concepts such as button, web page, URL. Couto, et al. [25] automated the extraction of requirements patterns based on stakeholders formalizing use case specification by using OWL inference capabilities to address typical implementation solutions. Dermeval, et al. [26] suggested that ontologies could be used for representing requirements and architectural knowledge and support reasoning through traceable links between them. This paper does not focus on bridging the gap between requirement and architectural design phases, but the concepts such as web page, button, other web UI components can be linked to artifacts in architectural design and subsequent phases, which help with traceability. Decker, et al. [27] represent use cases in a requirements document ontology to semi formalize the representation of actors interacting with the system through user story descriptions. Kang and Liang [9] developed a security ontology for software development, a model-driven approach, where security concerns play a role in the analysis, design, implementation, testing, and maintenance stages of the SDLC. Our approach is different from Kang and Liang [9] since we focus on applying security concerns to use case instead of representing use cases as ontologies for development.

Our work is different from the related literature in that it mapped specific flows in use cases to security concerns based on data and object properties.

VII. Conclusion and Future Work

This paper introduced a preliminary work on using an ontology framework during the early software development phases to identify security concerns based on use cases. We have manually and effectively created relationships between different use case concepts. These relationships have the potential to relate use case concepts to security concerns. Even though we

[7] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-14398

can use the ontology to identify security concerns, manually representing the user case in the ontology can be time inefficient for many use cases.

Also, this ontology currently works with a predefined set of rules for identifying security concerns. CVE provides information on many security attacks that are based on different CVE scenarios. For example, different parts of the use case can be exploited with XSS. New rules and queries can be developed to find where in the use case that could be affected XSS.

In future work, we intend to develop a web-based tool to automatically extract and populate the relevant information from use cases into the ontology. As a result, the tool will semi-automatically query ontology the parts of the use case that matches the security concerns rule. The work presented in this paper is a part of a larger project to help recommend relevant attack patterns as part of the security requirements process. We will evaluate the usability of the ontology framework in a user study with the participants in software engineering courses and the security requirements community.

ACKNOWLEDGMENT

This work is partially supported by NSF under grant CNS-1900187. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

REFERENCES

[1]. A. Souag, R. Mazo, C. Salinesi, and I. Comyn-Wattiau, "Reusable knowledge in security requirements engineering: a systematic mapping study," *Requirements Engineering*, vol. 21, pp. 251-283, 2016.

[2]. C. Schmitt and P. Liggesmeyer, "Getting grip on security requirements elicitation by structuring and reusing security requirements sources," *Complex Systems Informatics and Modeling Quarterly*, pp. 15-34, 2015.

[3]. S. Gärtner, T. Ruhroth, J. Bürger, K. Schneider, and J. Jürjens, "Maintaining requirements for long-living software systems by incorporating security knowledge," in *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, 2014, pp. 103-112.

[4]. C. Veres, J. Sampson, S. J. Bleistein, K. Cox, and J. Verner, "Using semantic technologies to enhance a requirements engineering approach for alignment of IT with business strategy," in *Complex, Intelligent and Software Intensive Systems, 2009. CISIS'09. International Conference on*, 2009, pp. 469-474.

[5]. A. Souag, C. Salinesi, R. Mazo, and I. Comyn-Wattiau, "A Security Ontology for Security Requirements Elicitation," in *ESSoS*, 2015, pp. 157-177.

[6]. H. Guan, H. Yang, and J. Wang, "An ontology-based approach to security pattern selection," *International Journal of Automation and Computing*, vol. 13, pp. 168-182, 2016.

[7]. T. OWASP, "10 2017," *OWASP Top 10 Application Security Risks–2017*, 2018.

[8]. N. F. Noy and D. L. McGuinness, "Ontology development 101: A guide to creating your first ontology," ed: Stanford knowledge systems laboratory technical report KSL-01-05 and Stanford medical informatics technical report SMI-2001-0880, Stanford, CA, 2001.

[9]. W. Kang and Y. Liang, "A security ontology with MDA for software development," in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2013 International Conference on*, 2013, pp. 67-74.

[10]. T. Yue, L. C. Briand, and Y. Labiche, "Facilitating the transition from use case models to analysis models: Approach and experiments," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, p. 5, 2013.

[11]. D. Ko, S. Kim, and S. Park, "Automatic recommendation to omitted steps in use case specification," *Requirements Engineering*, pp. 1-28, 2018.

[12]. J. Jurkiewicz and J. Nawrocki, "Automated events identification in use cases," *Information and Software Technology*, vol. 58, pp. 110-122, 2015.

[13]. S. Tena, D. Díez, P. Díaz, and I. Aedo, "Standardizing the narrative of use cases: A controlled vocabulary of web user tasks," *Information and Software Technology*, vol. 55, pp. 1580-1589, 2013.

[14]. P. Salini and S. Kanmani, "Security requirements engineering process for web applications," *Procedia engineering*, vol. 38, pp. 2799-2807, 2012.

[15]. I. O. f. Standardization, *ISO/IEC 27001: 2013: Information Technology--Security Techniques--Information Security Management Systems--Requirements*: International Organization for Standardization, 2013.

[16]. I. Horrocks, P. F. Patel-Schneider, and F. Van Harmelen, "From SHIQ and RDF to OWL: The making of a web ontology language," *Journal of web semantics*, vol. 1, pp. 7-26, 2003.

[17]. C. Welty, D. L. McGuinness, and M. K. Smith, "Owl web ontology language guide," *W3C recommendation, W3C (February 2004) http://www.w3. org/TR/2004/REC-owl-guide-20040210*, 2004.

[18]. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A semantic web rule language combining OWL and RuleML," *W3C Member submission*, vol. 21, p. 79, 2004.

[19]. M. A. Musen and T. the Protégé, "The Protégé Project: A Look Back and a Look Forward," *AI matters*, vol. 1, pp. 4-12, 2015.

[20]. G. De Giacomo and M. Lenzerini, "TBox and ABox reasoning in expressive description logics," *KR*, vol. 96, p. 10, 1996.

[21]. M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *The knowledge engineering review*, vol. 11, pp. 93-136, 1996.

[22]. M. Horridge and M. Musen, "Snap-SPARQL: a java framework for working with SPARQL and OWL," in *International Experiences and Directions Workshop on OWL*, 2015, pp. 154-165.

[23]. A. Rago, C. Marcos, and J. A. Diaz-Pace, "Uncovering quality-attribute concerns in use case specifications via early aspect mining," *Requirements Engineering*, vol. 18, pp. 67-84, 2013.

[24]. B. Wouters, D. Deridder, and E. Van Paesschen, "The use of ontologies as a backbone for use case management," in *European Conference on Object-Oriented Programming (ECOOP 2000), Workshop: Objects and Classifications, a natural convergence*, 2000.

[25]. R. Couto, A. N. Ribeiro, and J. C. Campos, "Application of ontologies in identifying requirements patterns in use cases," *arXiv preprint arXiv:1404.0850*, 2014.

[26]. D. Dermeval, J. Vilela, I. I. Bittencourt, J. Castro, S. Isotani, P. Brito, *et al.*, "Applications of ontologies in requirements engineering: a systematic review of the literature," *Requirements Engineering*, vol. 21, pp. 405-437, 2016.

[27]. B. Decker, E. Ras, J. Rech, B. Klein, and C. Hoecht, "Self-organized reuse of software engineering knowledge supported by semantic wikis," in *Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE)*, 2005, p. 76.