Ontology-Based Support for Security Requirements Specification Process

Olawande Daramola[1,2], Guttorm Sindre[2], and Thomas Moser[3]

[1] Department of Computer and Information Sciences
Covenant University, Ota, Nigeria
`Olawande.daramola@covenantuniversity.edu.ng`
[2] Department of Computer and Information Science
Norwegian University of Science and Technology (NTNU), Norway
`{wande,guttors}@idi.ntnu.no`
[3] Christian Doppler Laboratory for Software Engineering
Integration for Flexible Automation Systems
Vienna University of Technology, Austria
`thomas.moser@tuwien.ac.at`

Abstract. The security requirements specification (SRS) is an integral aspect of the development of secured information systems and entails the formal documentation of the security needs of a system in a correct and consistent way. However, in many cases there is lack of sufficiently experienced security experts or security requirements (SR) engineer within an organization, which limits the quality of SR that are specified. This paper presents an approach that leverages ontologies and requirements boilerplates in order to alleviate the effect of lack of highly experienced personnel for SRS. It also offers a credible starting point for the SRS process. A preliminary evaluation of the tool prototype – *ReqSec tool* - was used to demonstrate the approach and to confirm its usability to support the SRS process. The tool helps to reduce the amount of effort required, stimulate discovery of latent security threats, and enables the specification of good quality SR.

Keywords: security requirements, ontology, requirement boilerplates, information extraction, security threats.

1 Introduction

The increasing opportunities for systems integration, remote access, and sharing of resources across heterogeneous platforms by diverse software agents have made security requirements engineering (SRE) a major aspect of software system development in recent times. The Security requirements specification (SRS) process, which entails the formal documentation of identified security needs of a system, is an integral aspect of SRE [1]. However, there is a lack of sufficiently experienced security experts or security requirements engineers in many organizations, which limits the quality of SRS. SR becomes too vague or too specific in many cases due to the absence of experienced SR personnel [2, 3]. This scenario implies the need for a tool-based framework that is capable of supporting the SRS process. The framework will: 1) assist the requirements engineer (REng) in the identification of security threats, which is usually a manual procedure that depends largely on the expertise of human personnel; 2) stimulate the adoption of appropriate defence strategies to deal with the identified security threats; 3) enable the formulation of SR in a consistent way, eliminating ambiguity, and ensuring correctness of SR; and 4) reduce the effort needed for SRS by allowing the reuse of previously specified SR in subsequent instances. The aim is to assist the REng in the process of SRS so that the quality of SRS can be enhanced and effort reduced.

To achieve these objectives, we have integrated the use of ontologies and requirements boilerplates within a semantic framework-based tool to aid the REng personnel. The use of ontologies provides the necessary background knowledge, and domain knowledge that is required to identify security threats, and recommend appropriate countermeasures, while the requirements boilerplates provide a reusable template for writing SR in a consistent way in order to eliminate ambiguity. The uniqueness of our approach stems from the provision of a more elaborate procedure for supporting the SRS process relative to existing approaches because our approach: 1) enables identification of security threats; 2) provides recommendation of defence actions as countermeasure to identified security threats; and 3) enables pattern-based reuse of boilerplates when writing SR. The evaluation experiment that we conducted, reveal that our approach is usable to support SRS.

The rest of this paper is as follows. Section 2 gives an overview of background and related work, while Section 3 presents a detailed overview of our approach. Section 4 discusses the evaluation, results, and threats to validity. The paper is concluded in Section 5 with a discussion of further work.

2 Background and Related Work

In this section, we present an overview of ontology support for security requirements engineering (SRE), boilerplates for security requirements. Additionally, we discuss the related work.

2.1 Ontology Support for Security Requirements Engineering (SRE)

There is a lack of systematic processes for attaining software security [1], hence SRE attempts to add security considerations into software requirements engineering. SRE aims to integrate the security needs of a system particularly from the attacker's perspective into the software development process as early as possible. According to [3], SR objectives can be categorized as authentication, authorization, integrity, intrusion detection, non-repudiation, confidentiality, and auditing. Some well-known SRE approaches include Comprehensive, Lightweight Application Security Process (CLASP) [4], System Quality Requirements Engineering (SQUARE) [5], Common Criteria [6], Secure Tropos [7], and Misuse Case [8].

Ontologies as the semantic representation of the conceptualization of a domain have an important role to play in SRE. Research efforts on security ontologies such as [9, 10, 11] attest to this. In [12] the use of ontology was suggested as the solution to the problem of vaguely defined vocabularies among security practitioners.

According to [13], specific applications of ontologies to SRE include security taxonomies, general security Ontologies, specific security ontologies, security ontologies for Semantic Web, security ontologies for risk analysis, and ontologies for security requirements.

Generally, a good ontology will facilitate more effective reporting of incidents, sharing of information, and interoperable security collaborations among different organizations. Our proposed framework uses ontology to ensure the standardization of vocabulary in SRS, threat identification, and the recommendation of appropriate countermeasure to identified threats.

2.2 Boilerplates for Security Requirements

The notion of requirements boilerplates (RB) which stems originally from the work of [14], and subsequently applied in [15] enables the writing of requirements in a consistent manner. A requirement boilerplate is a pre-defined structural template for writing requirement statements. It imposes a uniform structure on the way requirements are written, by affording a level of expressivity akin to using of natural language, yet minimising ambiguity in requirement statements. The fixed parts of requirement boilerplate are reused when writing requirements, while the REng can fill in the parameter parts manually.

An example of a boilerplate taken from the webpage[1] is:

"BP2: **The <system> shall be able to <action> <entity>"**

Here, *BP2* is the label of this particular boilerplate. The terms in < > brackets are parameters where something must be filled in when the boilerplate is instantiated to a concrete requirement. The words that are outside brackets are the fixed syntax elements (FSE) that will be kept as-is when the boilerplate is instantiated. An example of an instantiation of this particular boilerplate, would be *"The payroll system shall be able to display login details of all its users"*. In this case <*action*> has been replaced by "display login details" and <*entity*> by "all its users". In some cases, several boilerplates may be combined to make precise and testable requirements, e.g. combining *BP2* with *BP37* **...at least <percentage> of the time** will yield the requirement *"The payroll system shall be able to display login details of all users at least 100% of the time"*.

Thus, the use of boilerplate will ensure that a unified structure and style of writing is used for requirements that pertain to specific classes of system function, capability, goals, or constraints. The FSE in each boilerplate will remain the same for all requirements that used a certain boilerplate. For instance, all who used BP2 + BP37 to specify that the system should be able to do something with some specific frequency,

[1] `http://www.requirementsboilerplates.org`

will now use phrases "shall be able to", "at least", "times per", rather than various other phrases that could have more or less the same meaning, e.g., "have the ability to", "be capable of", "a minimum of", "more than", "shots per", etc.

Firesmith in [16] identified four different types of defence against security threats, which can be used to assign specific security threats to the types of defence actions to counter them. These are:

(i) Prevention of malicious harm, security incident, security threats and security risks.
(ii) Detection of malicious attack, security incidents, security threats, and security risks.
(iii) Reaction to detected malicious attack.
(iv) Adaptation of system to avoid or minimize the negative consequences of the malicious harm, security incidents, security threats and security risks. This could also be in terms of recovery of system from attacks.

For each of the defence types, Firesmith also gave specific examples.

The examples in [16], could be the basis for some generic SR boilerplates, e.g.

SecBP1: **The** <*system*> **should [prevent | detect] at least** <*percentage*> **of** <*harm* | *incident* | *threat* | *risk*>
SecBP2: **Upon detection of** <*harm* | *incident* | *threat* | *risk*> **the system shall** <*action*>
SecBP11: **...of attacks with maximum duration** <*time unit*>
SecBP12: **...made by attackers with profile** <*attacker profile*>
SecBP21: **...at least** <*percentage*> **of the time**

Here, *SecBP11*, *SecBP12*, and *SecBP21* are parts that could be optionally concatenated with *SecBP1* or *SecBP2* respectively.

To use boilerplates for SRS in practical terms will entail the formulation of requirement boilerplates for the different aspects of security such as authentication, authorization, integrity, intrusion detection, non-repudiation, confidentiality, and auditing. Therefore, more boilerplates could be formulated, both as core parts and as attachments, but since boilerplates for security will vary for different domains, we cannot go into more detail here. However, experienced personnel must create the boilerplates prior to SRS as an upfront investment, while it should be updated periodically as new types of requirements emerge. By so doing, the boilerplate repository becomes an organisational asset for SRS that can be useful when there is paucity of experienced security REng personnel.

2.3 Related Work

We shall classify tool support for SRE into two broad categories – front-end tools and back-end tools. Front-end tools and approaches are those that facilitate the elicitation, modelling, and analysis of security threats in order to derive SR, while the back-end tools are those that help with the specification and validation of SR, and their integration with other requirements. Notable examples of front-end tools include: SecTro [17], - a CASE tool that supports automated modelling and analysis of security requirements based on Secure Tropos approach. The ST-Tool [18] supports the Secure

Tropos methodology. Its main goals are to support the translation of Secure Tropos models into formal specifications, and serve as a front-end tool for formal analysis of Secure Tropos models. The jMUCMNav (Java Use Case Map Navigator, [19]) editor is a modelling tool for Misuse Case Maps (MUCMs) in designing secure architectures for business processes. jUCMNav simply focuses on modelling for use case maps (UCM) and supports all UCM notations. Other front-end SRE tool that are worth mentioning are: SeaMonster [20, 21], and Surakasha security workbench [22].

Currently, there are more front-end tools than back-end tools for SRE. The SQUARE tool [5] is a back-end managerial tool that is designed to increase the quality of SRE process for the adopters of the SQUARE methodology. It support core SRE aspects such as definitions, searching, and addition of new terms, identification of security goals, assets and privacy goals, performing risk assessment, identifying threats, prioritizing requirements, traceability, and exporting of requirements to other requirements management tool. Similarly, the prototype tool - *ReqSec tool* – that we have developed is an eclipse-based back-end tool that supports SRS, and enables the integration of SR with other types of requirements. The unique feature of the ReqSec tool compared to other SRE back-end tools stems from its capability to facilitate automatic analysis of natural language requirements in order to assist the REng during SRS. It represents a first attempt to use semantic-based procedures for supporting both security threat identification and SRS. In the wider requirements engineering context, approaches such as [23] – ambiguity detection-, [24, 25] – requirement quality assessment-, are also based on natural language (NL) text analysis but did not use ontologies. The DODT [26] tool does not have a focus for SRE, but it bears similarity with our approach, because it combines the use of ontologies and boilerplates to enable semi-automatic transformation of NL requirements into boilerplate requirements. However, it can only ensure the correctness of requirements based on the underlying domain ontology, and the writing of boilerplate requirements. Our approach does more, in that it entails the discovery of latent security threats contained in NL descriptions, and the recommendation of probable defence actions that aids the formulation of semi-formal boilerplate SR. Hence, the novelty of our approach is the provision of a backend tool for SRE that will minimize effort needed for SRS, and offer a credible starting point for SRS, particularly in cases where there is paucity of experienced personnel.

3 Approach Overview

A high-level schematic overview of our approach is presented in Fig. 1. The process starts with input of description of the security threat scenario, which should be represented as a textual Misuse Case (TMUC) [8]. This is followed by identification of type of attack and required defence action through semantic text analysis of the TMUC, thereafter suggestion of boilerplates to be used to the REng, and finally specification of SR by the REng.

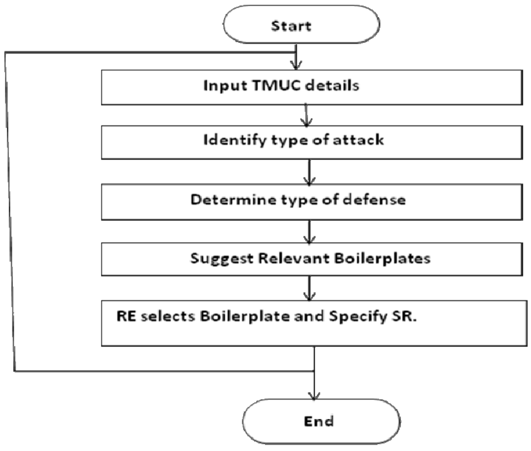

Start
Input TMUC details
Identify type of attack
Determine type of defense
Suggest Relevant Boilerplates
RE selects Boilerplate and Specify SR.
End

Fig. 1. Activity Workflow of the tool-supported SR Specification Process

3.1 Database Tampering - Example

In order to demonstrate how our tool-supported framework can be applied, we hereby consider the example of a security threat description of database tampering scenario. The detail of the scenario is presented in Table 1 using a TMUC template.

Table 1. TMUC for Database Tampering Case

Code: QC1	
Misuse Case Title	Tamper with database by web query manipulation
Name of System	Web Query System
Summary	A crook manipulates the web query, submitted from a search form, to update or delete information, or to reveal confidential information;
Basic Path	The crook provides some values on a product search form and submits. The system displays the product(s) matching the query. The crook alters the submitted URL, introducing a query error, and resubmits. The query fails and the system displays the database error message to the crook, revealing more about the database structure. The crook alters the query further, for instance adding a nested query to reveal secret data or update or delete data, and submits. The system executes the altered query, changing the database or revealing content that should have been secret.
Alternative Path	The crook does not alter the URL in the address window, but introduces errors or nested queries directly into form input fields.

Input TMC Details

The TMUC template [8] has two core aspects namely the basic path, and the alternative path. The basic path describes the security threat scenario that could be used by an attacker to cause harm to a system, while the alternative path specifies the other options that may be explored by an attacker or user with malicious intent. These two aspects together with the TMUC summary provide the key inputs used to identify the type of attack, and required defence for the system.

Identify Type of Attack
We used information extraction technique to identify the type of attack described by a TMUC template. The textual input are semantically analysed in order to identify and extract the most important (theme) words that have security implications. The basic threat ontology (BTO) (see Fig. 2), WordNet ontology, and the domain ontology (DO) are used to do this. A theme word can be the subject of a sentence (noun), or an action word (verbs) or a word collocation that connote a security threat to a system when it has been analysed. Core natural language processing algorithms for tokenization, parts-of-speech tagging, syntax parsing, morphological analysis, and ontology-based inferencing were used to achieve this task.

Determine the Type of Defence Using the Basic Threat Ontology (BTO)
The BTO contains a mapping of different kinds of security threats to specific defence actions based on information that was gathered from the literature and a number of existing security ontologies. The BTO is a major investment and a core knowledge infrastructure of the framework. The defence actions are the ones proposed by Firesmith in [16]. We reused all the essential aspects of the threat description in Security Ontology [11] as foundation for developing the BTO, which included some additional concepts. The BTO has a total of 98 classes, 46 restrictions and 9 object properties. The key object properties include *hasDefense* – which associates a threat with a specific defensive action, *hasThreat* – which associate a threat with an asset, *isThreathenedBy* – inverse of hasThreat, *isThreatTo*, *isSameAs*, - which describes equivalent concepts. Each security threat in the BTO was mapped to one or more defence actions (viz. detect, prevent, adapt, react, recover) using the *hasDefense* object property. Figure 2, presents a view of the BTO illustrating how specific types of attacks have been mapped to corresponding defence actions. The knowledge contained in the BTO is used for automatic recommendation of appropriate defence actions when a particular type of attack has been identified from the TMUC input details. The Pellet OWL Descriptive Logics (DL) reasoner was used as the ontology reasoning engine for the BTO.

Suggestion of Relevant Boilerplates
The information extraction process generates a set of recommendations comprising a pair of defence-action and attack-type. Fig. 3 shows the recommended pairs for the Database Tampering example. The recommended pairs are extracted directly from the BTO after the semantic analysis of the TMUC. Ontology reasoning and other semantic capability were facilitated using Stanford NLP toolkit[2], Word Net java API, and the Jena semantic framework[3]. Per-time, the REng will have to select a specific <defence-action, attack-type> pair from the list of recommendations, and appropriate

[2] `http://nlp.stanford.edu/software/lex-parser.shtml`
[3] `http://jena.sourceforge.net/`

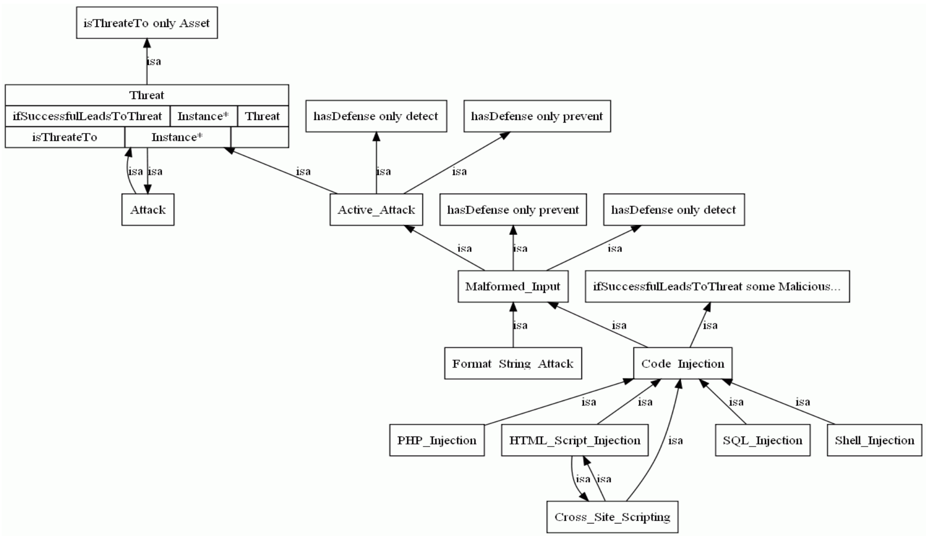
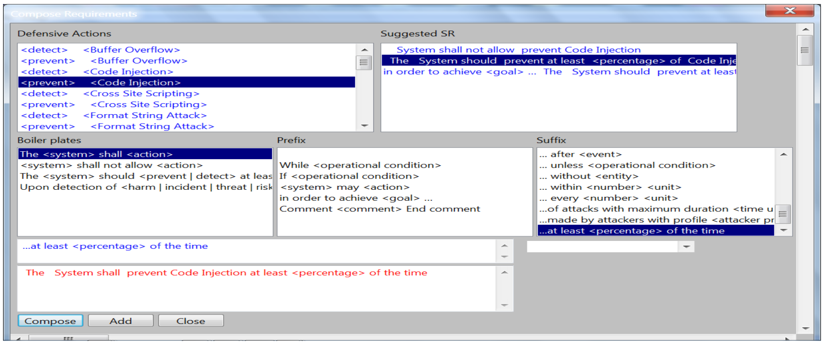


Fig. 2. A view of the description of Malformed Input threat in the BTO using OntoViz

boilerplate, prefix and suffix, from the boilerplate repository to formulate a security requirement automatically. However, the tool is able to learn by keeping track of the combinations of <defence-action, attack-type> pairs and boilerplate patterns that tend to go together based on user's preferences. Subsequently, once a <defence-action, attack-type> pair is selected, the tool automatically displays a list of fully formulated boilerplate SR for the user to select from. This way, the fixed syntax elements (FSE) of the selected boilerplate are reused, while the selected <defence-action, attack-type> substitutes the <action> placeholder in the selected boilerplate. The REng can then fill in the remaining part of the boilerplate requirements that require specific data to complete the formulation of the SR (see Fig. 3).


Compose Requirements

Defensive Actions

```
<detect>    <Buffer Overflow>
<prevent>    <Buffer Overflow>
<detect>    <Code Injection>
<prevent>    <Code Injection>
<detect>    <Cross Site Scripting>
<prevent>    <Cross Site Scripting>
<detect>    <Format String Attack>
<prevent>    <Format String Attack>
```

Suggested SR

```
 System shall not allow  prevent Code Injection
 The  System should  prevent at least  <percentage> of  Code Inje
in order to achieve <goal> ...  The  System should  prevent at least
```

Boiler plates

```
The <system> shall <action>
<system> shall not allow <action>
The <system> should <prevent | detect> at leas
Upon detection of <harm | incident | threat | risk
```

Prefix

```

While <operational condition>
If <operational condition>
<system> may <action>
in order to achieve <goal> ...
Comment <comment> End comment
```

Suffix

```
... after <event>
... unless <operational condition>
... without <entity>
... within <number> <unit>
... every <number> <unit>
...of attacks with maximum duration <time u
...made by attackers with profile <attacker pr
...at least <percentage> of the time
```

```
...at least <percentage> of the time
```

```
The  System shall  prevent Code Injection at least <percentage> of the time
```

Fig. 3. A snapshot of suggestions for database tampering from the tool

4 Evaluation

We conducted a preliminary evaluation of our approach by using a controlled experiment with seven subjects. The subjects were Master degree students of software engineering of NTNU, Norway, who volunteered to participate in the experiment. The aim was to assess the usability of the tool for supporting SRS. The participants were paid for taking part in the experiment.

Background of Participants
The response to a pre-experiment questionnaire revealed that the participants had good background knowledge in the specific areas such as system security, requirements engineering, ontology, and boilerplates that pertain to the experiment. They have all taken two relevant courses – software security and requirements engineering and testing – in the department – IDI / NTNU. In addition, the majority of the subjects also claim to have some industrial work experience.

Evaluation Procedure
The participants were asked to use the *Reqsec tool*[3] during a controlled experiment that lasted for 1.5 hours. They were presented with four security threat scenarios, and asked to formulate SR for each case by using the tool. All the participants performed the same task at any given time during the experiment. The participants were given a five minutes tutorial on the use of the tool[4] before they commenced the experiment. They were required to assess the tool along six dimensions - *perceived usefulness (PU), perceived ease of use (PEOU), intention to use (ITU), reuse (reu), accuracy (acc), and serendipity (sere)* - through a post-experiment questionnaire. The mean score out of a maximum of 5.0 for each of the six dimensions are shown in Table 1.

4.1 Results

The analysis of the results from the post-experiment questionnaire revealed that the tool had its highest mean rating in the aspects of *perceived ease of use (PU)*, and *serendipity (sere)* – the users acknowledged that the tool offered suggestions that they had not thought about originally. The tool also had good rating in other aspects such as *reuse*, *accuracy*, and *intention to use*. All the participants stated emphatically that they would use the tool.

In the free comments feedback section of the questionnaire, the participants revealed a positive general perception of the tool as potentially viable to support SRS, and admitted their willingness to use it. Most agreed that the tool is easy to use, and capable of assisting a REng. A few of them were particularly impressed that the tool enabled them to write security requirements that they did not think about initially until when they saw the suggestions from the tool. They all agreed that although the tool offers useful support for SRS, it cannot be solely relied upon. This is because there were occasions when the tool failed to suggest certain expected options. Some of them advised that the tool would perform better if the quality of the underlining

[3] `https://www.idi.ntnu.no/~wande`
[4] `https://www.idi.ntnu.no/~wande/Guide_for_Reqsec_Tool.htm`

ontology is improved. They also mentioned a number of areas that should to be improved in the tool. This includes the fact that 1) the tool's interface did not scale well on the MacOs systems compared to Windows; and 2) the need to be able to save the requirements that pertain to a TMUC all at once in the repository and not one at a time. We agree with the observations of the participants and would seek to revise the subsequent version of the tool based on the observations by participants.

Generally, the result of the evaluation demonstrates the potential of the tool to first, simplify, and significantly aid the REng during the SRS process, particularly when the REng is not highly experienced. Second, facilitate a reduction in the effort expended on SRS, particularly as the process progresses. Third, ensure that correct terms are used when formulating SR, and in a consistent way without ambiguity. However, our inspection of the specified security requirements revealed consistency in the use of language and pattern of expression in formulated SR that pertain to same security threat scenario by different individuals, which is mainly due to the use of boilerplates and ontologies.

Table 2. Mean score rating for Tool Assessment

Metric	Mean	Std
PU	3	0.433013
PEOU	3.714286	0.698638
ITU	3.357143	0.481039
Reu	3.214286	0.393398
Acc	3.285714	0.95119
Sere	4	0.816497

4.2 Lessons Learned

Our experience from the evaluation emphasised the need for high quality underlying ontologies – BTO, DO - and the boilerplate repository. Hence, an upfront and crucial investment is to ensure that good quality BTO, DO and a rich boilerplate repository are available at the onset of the tool. In order to cater to this, the tool comes pre-loaded with the BTO and the boilerplate repository as basic artefacts, while a DO can be imported into the tool. Also, provision was made to ensure the evolution of the BTO, DO, and boilerplate repository with time. To do this, we have made it possible to continually revise the ontologies BTO, DO, and boilerplate repository from within the tool's environment. The tool includes an ontology management module that allows the addition of new concepts, properties, and axioms to an existing ontology, while the boilerplate management module allows the boilerplate repository to be updated. Thus, the tool can be customised, and adapted to cater for future emerging requirements.

4.3 Evaluation Threats

Ordinarily, an industrial case study would give a different perspective to the evaluation of the tool and the quality of tool support. However, the subjects used for the experiment are sufficiently knowledgeable in the relevant areas such as requirements engineering, system security, ontologies, and requirements boilerplates having taken taught courses in these areas. This makes them suitable as reasonable substitutes for real experts in a preliminary evaluation. Also, the evaluation was performed with only seven users, but although the statistical significance is reduced, the results are indicative of the acceptance of the approach evaluated. Moreover, our objective is to assess the potential usability of the tool to support SRS. Evidence in literature suggests that a minimum of 5 subjects are sufficient to get a valid opinion on the usability of a tool [27].

Another perspective to the evaluation could be to evaluate the tool alongside other tools or to compare its performance with humans, either of, which could also lead to a different result compared to what we have reported. However, comparative evaluation with other tools is not attractive as at now because, hardly could we find any other tool that have the same focus, and is set out to do exactly a similar thing as we envisioned. The option to compare the tool capacity with human is a possibility for the future, after this preliminary evaluation.

5 Conclusion

In this paper, we have presented the notion of ontology-support for security requirements specification. Our approach employs a tool-based framework that uses a combination of ontologies and boilerplates to aid a requirements engineer in the process of security threat identification and eventual formulation of quality SR. It provides the attendant benefits of reducing the effort need for the SRS process, and offers a good starting point in cases when sufficiently experienced REng may not be available. The preliminary evaluation of the approach confirms that it is viable and usable for supporting SRS. In future work, we will conduct a more elaborate evaluation by using industrial case studies to further validate the approach. Also, we shall seek means to further improve the performance of the tool, and extend the concepts to the aspect of safety.

Acknowledgment. The Norwegian Research Council through the ReqSec project, Norway, has supported this work while the first author of this paper was a Research Scientist at NTNU, Norway. This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

References

1. Rushby, J.: Security Requirements Specifications: How and What? Symposium on Requirements Engineering for Information Security (SREIS), Indianapolis (2001)
2. Firesmith, D.: Specifying Reusable Security Requirements. Journal of Object Technology 3(1), 61–75 (2004)
3. Chandrabrose, A.: Alagarsami: Security Requirements Engineering – A Strategic Approach. International Journal of Computer Applications 13(3), 25–32 (2011)

4. Viega, J.: The CLASP Application Security Process. Training Manual, vol. 1(1). Secure Software Inc. (2005)
5. Mead, N., Stehney, T.: Security quality requirements engineering (SQUARE) methodology. In: Proceedings of International Conference on Software Engineering for Secure Systems (SESS 2005), pp. 1–5 (2005)
6. Common Criteria Implementation Board. Common Criteria for Information Technology Security Evaluation, Part 2: Security Functional Requirements (1999)
7. Mouratidis, H., Giorgini, P.: Secure Tropos: A security-oriented extension of the Tropos methodology. International Journal of Software Engineering and Knowledge Engineering 17(2), 285–309 (2004)
8. Sindre, G., Opdahl, A.: Eliciting Security Requirements with Misuse Cases. Requirements Engineering 10(1), 34–44 (2005)
9. Fenz, S., Ekelhart, A.: Formalizing information security knowledge. In: 4th International Symposium on Information, Computer, and Communications Security (ASIACCS 2009), pp. 183–194 (2009)
10. Kim, A., Luo, J., Kang, M.: Security Ontology for Annotating Resources. In: 4th International Conference on Ontologies, Databases, and Applications of Semantics, ODBASE 2005 (2005)
11. Herzog, A., Shahmehri, N., Duma, C.: An Ontology of Information Security. International Journal of Information Security 1(4), 1–23 (2007)
12. Donner, M.: Toward a Security Ontology. IEEE Security and Privacy (2003)
13. Souag, A., Salinesi, C., Wattiau, I.: Ontologies for Security Requirements: A Literature Survey and Classification. In: WISSE 2012 in Conjunction with 24th International Conference on Advanced Information Systems Engineering (CAiSE 2012), pp. 8 pages (June 2012)
14. Hull, E., Jackson, K., Dick, J.: Requirements Engineering. Springer (2004)
15. Daramola, O., Stålhane, T., Sindre, G., Omoronyia, I.: Enabling Hazard Identification from Requirements and Reuse-Oriented HAZOP Analysis. In: Proceeding of 4th International Workshop on Managing Requirements Knowledge, pp. 3–11. IEEE Press (2011)
16. Firesmith, D.: A Taxonomy of Security-Related Requirements. In: Proceedings of the International Workshop on High Assurance Systems (RHAS 2005), Paris, France (2005)
17. Pavlidis, M., Islam, S., Mouratidis, H.: A CASE Tool to Support Automated Modelling and Analysis of Security Requirements, Based on Secure Tropos. In: Nurcan, S. (ed.) CAiSE Forum 2011. LNBIP, vol. 107, pp. 95–109. Springer, Heidelberg (2012)
18. Giorgini, P., Massacci, F., Mylopoulos, J., Siena, A., Zannone, N.: ST-Tool: A CASE Tool for Modeling and Analyzing Trust Requirements. In: Herrmann, P., Issarny, V., Shiu, S.C.K. (eds.) iTrust 2005. LNCS, vol. 3477, pp. 415–419. Springer, Heidelberg (2005)
19. Bizhanzadeh, Y., Karpati, P.: jMUCMNav: an Editor for Misuse Case Maps. In: First Int. Workshop on Alignment of Business Process and Security Modelling (ABPSM 2011), Riga, Latvia (2011)
20. Tøndel, I.A., Jensen, J., Røstad, L.: Combining misuse cases with attack trees and security activity models. In: Proc. ARES 2010, pp. 438–445 (2010)
21. http://sourceforge.net/apps/mediawiki/seamonster/
22. Maurya, S., Jangam, E., Talukder, M., Pais, A.R.S.: A security designers' work-bench. In: Proc. Hack. in 2009, pp. 59–66 (2009)
23. Gleich, B., Creighton, O., Kof, L.: Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources. In: Wieringa, R., Persson, A. (eds.) REFSQ 2010. LNCS, vol. 6182, pp. 218–232. Springer, Heidelberg (2010)

24. Wilson, W., Rosenberg, L., Hyatt, L.: Automated Analysis of Requirement Specifications. In: Proceedings of the International Conference on Software Engineering (ICSE 1997), pp. 161–171 (1997)
25. Fabrini, F., Fussani, M., Gnesi, S., Lami, G.: An Automatic Quality Evaluation for Natural Language Requirements. In: Proceeding of the Seventh International Workshop on Requirements Engineering Foundation for Software REFSQ 2001, Interlaken, Switzerland, pp. 150–164 (2001)
26. Farfeleder, S., Moser, T., Krall, A., Stålhane, T., Zojer, H., Panis, C.: DODT: Increasing Requirements Formalism using Domain Ontologies for Improved Embedded Systems Development. In: Proceedings of 14th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS 2011), pp. 1–4 (2011)
27. Nielsen, J., Landauer, T.: A mathematical model of the finding of usability problems. In: Proceedings of ACM INTERCHI 1993 Conference, pp. 206–213 (1993)