Capturing Security Requirements Using Essential Use Cases (EUCs)

Syazwani Yahya[1], Massila Kamalrudin[1,**], Safiah Sidek[1], and John Grundy[2]

[1] Innovative Software System and Services Group,
Universiti Teknikal Malaysia Melaka, Malaysia
[2] Faculty of Science, Engineering and Technology
Swinburne University of Technology
Melbourne, Australia
{massila,safiahsidek}@utem.edu.my, syazwaniyahya13@gmail.com,
jgrundy@swin.edu.au

Abstract. Capturing security requirements is a complex process, but it is crucial to the success of a secure software product. Hence, requirements engineers need to have security knowledge when eliciting and analyzing the security requirements from business requirements. However, the majority of requirements engineers lack such knowledge and skills, and they face difficulties to capture and understand many security terms and issues. This results in capturing inaccurate, inconsistent and incomplete security requirements that in turn may lead to insecure software systems. In this paper, we describe a new approach of capturing security requirements using an extended Essential Use Cases (EUCs) model. This approach enhances the process of capturing and analyzing security requirements to produce accurate and complete requirements. We have evaluated our prototype tool using usability testing and assessment of the quality of our generated EUC security patterns by security engineering experts.

Keywords: Software Engineering, Requirements Capturing, Security Requirements, Secure Software Development, Essential Use Case (EUC).

1 Introduction

There is an increasing need to look at the cost, reliability and safety of software systems. With the increase of threats and vulnerabilities in many software systems, security issues involving software have become widespread, frequent and serious. We believe that enumerating accurate security requirements can help system architects and security engineers to develop realistic and meaningful secure software [1]. Security requirements elicitation is usually conducted during the early phase of the system life cycle. Often these are only generic lists of security mechanisms, such as password protection, firewalls, virus detection tools and SSL layer (for cryptographically protecting communications) [1, 2] [12]. However, these security requirements often do

* Corresponding Author.

D. Zowghi and Z. Jin (Eds.): APRES 2014, CCIS 432, pp. 16–30, 2014.

not present a complete solution to the security problems of the target application under development. It is crucial for software engineers to accurately capture the essential security mechanisms (such as access control) and implement the correct design solutions for security (such as robust design and good technology choices) that makes software attacks much more difficult. In our experience, we have found that security requirements elicitation and analysis necessary for a better set of security requirements seldom happens. According to Salini [12], even if it is done, the security requirements are often developed independently from the rest of the requirements engineering activities: They are not integrated into the mainstream of the requirements engineering activities. As a consequence, security requirements that are specific to the application of software or system and the protection of essential services and assets are often neglected. A lot of requirements engineering research and practice have tried to address the security capabilities that a software or system should provide. However, they have limited focus since they tend to describe design solution in terms of protection mechanisms only. They lack of making declarative propositions [12] with regards to the required level of protection that can be accurately established by capturing the correct security requirements in the first place.

A lot of attention has been given to the functional requirements of the system from the user's view, whilst less attention is given to security requirements. [6][12]. Many practices do not tackle security requirements at all, but rather focus on the implementation mechanisms intended to satisfy unstated requirements and assumptions. As a result, security requirements that are specific to the system and that provide protection of essential services and assets are often neglected. This can cause substantial security problems at a later stage [2, 3]. In practice, when capturing security requirements from clients, requirements engineers frequently use some forms of natural language, written either by clients or themselves. This forms a human-centric representation of the requirements accessible to both the engineers and clients. However, due to the ambiguities and complexities of the natural language and the process of capture, these requirements often have inconsistencies, redundancies, incompleteness and omissions which can lead to the development of inaccurate secure software. Our research described in this paper introduces a new, more effective approach using semi-formal models called Essential Use Cases (EUCs) that support requirements engineers in capturing security requirements. This study is an extension of our earlier work [34] [36] to provide better supports for the capturing of security requirements that can enhance the correctness and completeness of the captured security requirements.

This study contributes to the enhancement of the quality of software intended to be developed. The essence of the approach is to support capturing security elements from the normal business requirements expressed in natural language. To allow requirements engineers and stakeholders to detect security requirements, we adopted the concept of essential interactions patterns and essential use case patterns. We highlighted the potential of this tool for quality security requirements by annotating a visual, semiformal model of the security requirements and normal business requirements depicted in our support tool. We evaluated the usability of our prototype tool using a survey conducted with a sample of selected end-users. An evaluation of our security requirements patterns by experts was also conducted.

2 Background

The Essential Use Case (EUC) approach was developed by Constantine and Lockwood [37]. EUCs are designed to resolve problems which occur in conventional Use Case modeling and they have important benefits over that approach [41]. An EUC is defined as a "structured narrative, expressed in a language of the application domain and of users, comprising a simplified, generalized, abstract, technology free and independent description of one task or interaction that is complete, meaningful, and well-defined from the point of view of users in some role or roles in relation to a system and that embodies the purpose or intentions underlying the interaction" [37]. An EUC is thus a form of dialogue between a user and a system which supports better communication between the developers and the stakeholders. An EUC is shorter and simpler as compared to a conventional use case since it comprises an abstraction of only essential steps and the user's intrinsic interest. An EUC aims to identify "what the system must do" without being concerned on "how it should be done".

EUCs are made up of a set of organized "abstract interactions" and EUCs extracted from natural language specifications can be compared against templates of "interaction patterns" to detect requirements quality problems. Requirements engineers need to derive appropriate essential interactions from the requirements at a correct level of abstraction. Biddle et al. [42] and Kamalrudin et al. [5] found that almost all users have problems defining the right level of abstraction and exerted that the abstraction process to be time consuming. These problems are some of the reasons why it is difficult to check security requirements for consistency and completeness. In this case, we anticipate that a Security Essential Interaction Library can mitigate these problems. This library consists of important key phrases (security essential interactions) and mappings to appropriate essential security requirements (security abstract interactions).

A key reason for choosing the EUC model is that it lends itself to a deeper analysis that enables the identification of security requirements by extracting from the normal business requirements. Once a EUC model has been extracted, it can be compared against a pattern in our SecEUC Interaction Pattern Library.

3 Motivation

Many studies have found that most software engineers have poor training in eliciting, analyzing, and specifying the security requirements. This is due to a considerable lack of security knowledge [1] [15] [16] [17] [18]. New security challenges are growing along with today's complexity and interoperability software systems development. Requirements are provided by a variety of project partners; thus, the specifications are voluminous and contain many requirements. Further, the process of manually eliciting requirements is tedious [3]. To correctly capture security requirements, good skills and knowledge in both the requirements and security areas are required. Shielding security loopholes and establishing correct and accurate security requirements are considered to be a difficult task. Yet, this essential element is taken for granted by

many. It is important for requirements engineers to understand that security requirements are more than just dealing with security solutions that provide strong passwords, configure SSL, or validate user input. It involves a process of accurately capturing the right security controls for what the applications and business really needs. However, requirements engineers often fail to pay sufficient attention to security concerns, treating them as non-functional requirements [4]. The majority of software projects deal with the security when the system has already been designed and sometimes even when it has been put into operation. In extreme cases, the actual security requirements themselves are never well understood [4]. Requirements engineers without sufficient experience in security face the risk of over-looking security requirements leading to security vulnerabilities that are easily exploited [3]. It is widely known that security requirements need to be considered at the early stage of software development.

Recognizing the importance of security requirements in achieving a secure software development, Microsoft has adopted a systematic security assurance process, the Security Development Lifecycle (SDL). As a company-wide initiative and a mandatory policy since 2004, the SDL has a significant role in embedding security and privacy in the software and culture at Microsoft. During the software development life cycle (SDLC), security requirements are elicited at the most early phase, which is at the requirement phase, as shown in Figure 1 [13]:

Fig. 1. The Microsoft Security Development Lifecycle – Simplified [13]

In a software organization, it is common to have a project team that consists of requirements engineers and security engineers. The primary responsibilities of requirements engineers or system analysts are to gather, analyze, document and validate the needs of the project stakeholders [14]. They are responsible at the requirement phase which is to capture security requirements from clients. Security engineers, on the other hand are responsible for designing, developing and deploying security related systems and security in systems. Their responsibilities and skills can be very specific such as designing a hardware security appliance [19]. The task of a security engineer is usually centered at the implementation or design phase.

Although both engineers have complementary responsibilities in capturing requirements, they do not communicate effectively with each other; hence, there is a

lack of integration on the work done between them. This condition can lead to inconsistency and incorrectness of the developed software and it fails to fulfill the needs of the stakeholders. Additionally, the existing standard, such as the Common Criteria (ISO) has been identified as extensive, complex and difficult to comprehend by requirements engineers [27]. The existing techniques, such as interviews and brainstorming are found to be time consuming and fail to accurately identify security requirements. In this case, captured security requirements using the existing standards and techniques are prone to be inaccurate, inconsistent and incomplete, and this can lead to insecure software systems.

4 Our Approach

We explored the use of semi-formal model Essential Use Cases (EUCs) to develop a new approach for capturing security requirements. We automate the capturing process of security requirements from the business requirements using (EUCs) model. Further, a rapid prototyping approach was adopted to ensure the production of accurate and secure software. Next, pattern libraries called Security Essential Use Cases (SecEUC), Security Essential Interactions (SecEI) and Security Controls Patterns (SecCtrl) library were developed to assist the capturing process for security requirements. This is a lightweight approach that allows requirements engineers to identify and capture the security requirements and keep them consistent within the business requirements. The following section describes the pattern library for capturing security requirements.

4.1 SecEUC Pattern Libraries

Our security pattern library consists of three library patterns, which are the Security Essential Use Cases (SecEUC), Security Essential Interaction (SecEI), and Security Controls (SecCtrl) patterns. The SecEUC library patterns which is based on EUCs was generated from the normal business requirements, while the SecEI library patterns is based on the essential interactions found in the textual requirements related to security elements. The development of the SecEUC patterns was adapted from the works of Kamalrudin et al. 2011 [34] [36]. Through the extraction process, phrases from the textual natural language requirements were analyzed and matched to the essential interaction pattern library to find an appropriate abstract interaction. The abstract interactions associated with security are called SecEUC. Essential Interactions that contain the security elements are called SecEI.

The identification of associated security elements are based on the definitions from the basic security services [38]. It was found that multiple SecEI are associated with one SecEUC. For example, the essential interaction “key in username and password” and “log in” were identified as security related and they were mapped to a SecEUC “identify self”. This is because all of them have the attributes of security. Other examples of the pattern library are shown in Table 1.We then designed our SecCtrl library patterns based on basic security services [38], such as the access control (authorization), authentication (integrity), confidentiality (privacy), availability and accountability (non-repudiation). The SecCtrl was developed for the purpose of

mapping it to the prototype generation and providing the mandatory security controls. This pattern library helped us to identify the security controls that are relevant to a particular SecEUC. The association between the SecEUC and SecCtrl is that one SecEUC can have one or more than one SecCtrl. For example, "Identify Self" of SecEuc was mapped to "Authentication" and "Authorization" SecCtrl. Other examples of the pattern library are shown in Table 1.

Table 1. Examples of our security-oriented EUC pattern libraries

SecEI	SecEUC	SecCtrl
Check password	Identify Self	Authentication Authorization
Check username		
Verify username		
Make payment	Make payment	Authentication Transaction
Complete payment form		

4.2 Using Our Approach

Figure 2 shows the overview of our approach that enhances the process of capturing security requirements. The process of our approach begins after the requirement engineer gathered the requirements from the stakeholders. The collected requirements are in the forms of textual natural language requirements. The followings are the sequence of the process.

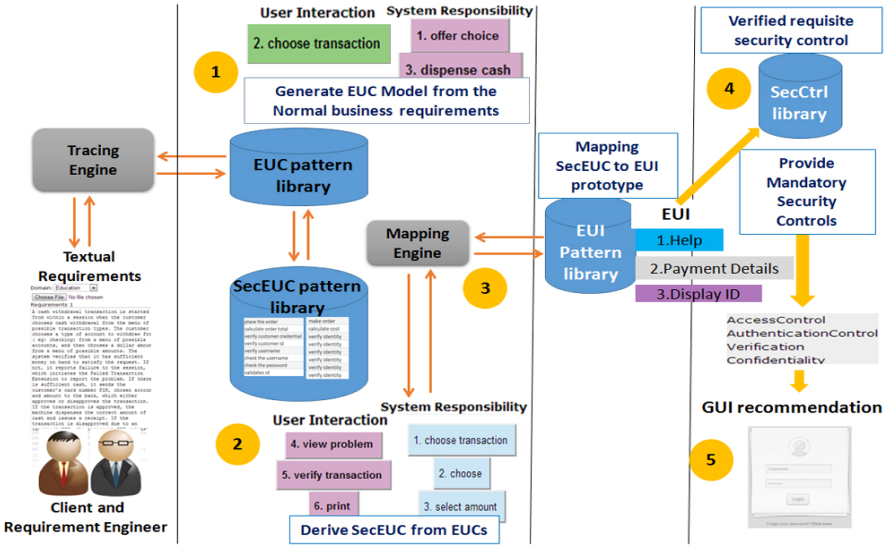

Fig. 2. Overview of our approach

The process starts when the textual requirements are analyzed and traced to the EUCs patterns library for appropriate abstract interaction in a form of EUC model (1). Then, SecEUC are derived from the generated EUC Models based on the categorization of their attribute related to the security element as defined in the SecEUC pattern library (2). Each SecEUC is mapped to EUI pattern library (3) for the generation of abstract prototype in a form of EUI model. Then, each EUI model is verified with a defined mandatory security control in the SecCtrl library patterns (4). Next, a recommendation of graphical user interfaces (GUI) is provided to visualize the security requirements based on the generated SecEUC (5). This helps to ensure the consistency and correctness of the captured security requirements with the original business requirements provided by the end-user.

5 Tool Support and Usage Example

5.1 SecMEReq : Prototype Tool

We have developed a prototype tool to support our EUC-based requirements capture and analysis process, an extension of our earlier MEReq [7] tool. Figure 3 shows our extended version of MEReq, called SecMEReq. The tool allows requirements engineers to automate the elicitation process for capturing security requirements. The selected phrases in the textual requirements show the resulting extracted security essential interactions. Meanwhile, the selected essential interactions show the sources from which the textual natural language phrases were derived. This provides a traceability support mechanism between the textual natural language requirements and the derived security EUC models. Engineers can then modify the generated security EUC model and/or the original textual natural language requirements. This includes adding phrases and interactions, re-ordering phrases and interactions, uploading and re-uploading requirements, deleting phrases and interactions and modifying phrases and interactions descriptive texts. Users (engineers) are also allowed to re-extract the essential interactions and associated traceability links. In this case, engineers need to have a basic understanding of the Essential Use Case concept and methodology only. To demonstrate, our tool key features, user scenario and figure of the tool support are provided as below:

Nancy, a requirements engineer, would like to validate the security requirements which has a mixture of the business and security requirements, which she has collected from a client, Nick, who is a car rental information manager. To do this, as shown in Figure 3, she types the requirements in a form of user scenario or copies them from an existing file on the textual editor (1). Once she has finished typing or copying the requirements, the tool generates the model of the essential requirements (abstract interactions) and the screen will show the EUC models containing the user interaction and system responsibility side by side to the chosen requirements (2). On the same display screen, she verifies the list of abstract interactions provided by the tool as shown in figure 4. From the generated EUC, she then captures the security requirements from the business requirements in a form of SecEUC which is presented in the green color boxes. Further, she checks the consistency and dependencies

between the SecEUC components and the SecEI by performing a trace back using the "capture SecEI" (3) event handler. From here, she could verify the consistency between the captured security requirements with the original textual requirements. At this stage, the associated SecEI are highlighted with yellow colors at the textual requirements (3A). In order to further validate her captured security requirements and to ease the discussion process with Nick, she has the tool map the SecEUC model to the low-fidelity prototype - EUI prototype (3B). As shown in Fig 3(3B), the EUI model that has the relation with security is also colored in green. From the EUI prototype, the tool then provides her with a set of mandatory security control for the captured SecEUC (4). In order to visualize the captured security requirements, she then has the tool that translates the EUI prototype to a more concrete UI view (5). From here, she then verify the consistency and the correctness of the captured security requirements with the client-stakeholder.

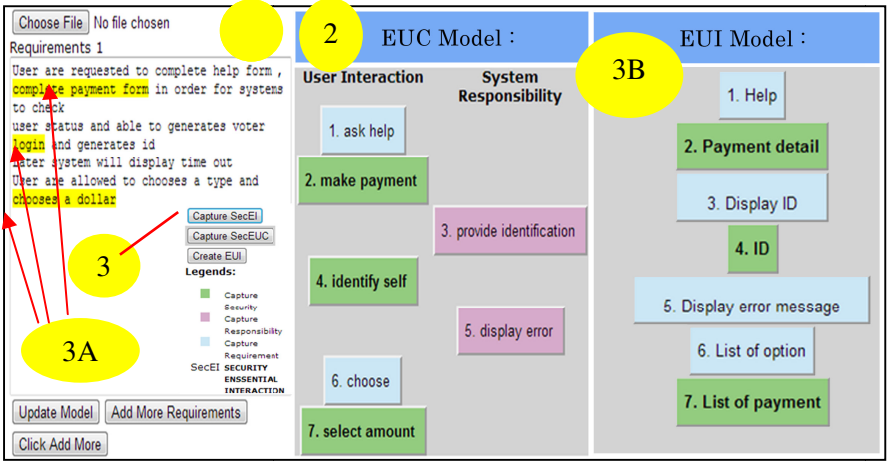

Fig. 3. Requirements Example and a SecMereq Usage Example

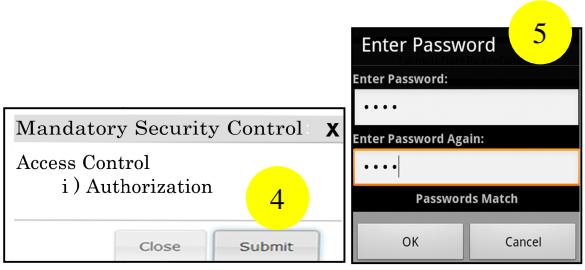

Fig. 4. SecMereq Tool Support in use

5.2 Tool Architecture

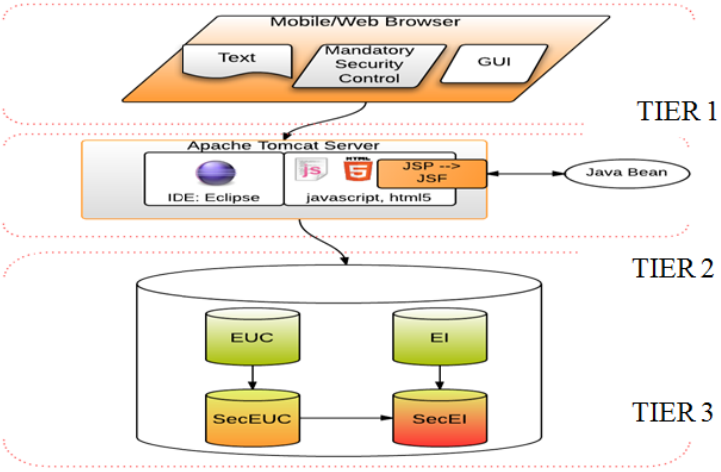

Fig. 5. A High-level architecture of SecMereq

We have enhanced our Mereq [7] tools by adding a new module and functions to capture security requirements. Figure 5 shows the high-level architecture of the prototype tool that consists of three tiers. The first tier contains the front-end that handles the interaction with users. Users are able to view the prototype tool either from a mobile or a web browser. A textual documentation of the requirement is inserted and the mandatory security controls are provided along with the recommendations for suitable graphical user interfaces (GUI). The middle-tier contains the rules for the processing information. A dynamic content processing and generation takes place at this level. Apache tomcat servers are utilized as the middleware or platform for the development at this stage. To build the Java applications, Eclipse Juno was selected for the IDE as it is recognized to provide a superior Java editing with validation and code assistance. Java Server Pages (JSP) allows writing a text using client's languages, which are the JavaScript and HTML5. Tier 3 manages the access to the database that stores the patterns library: the essential use cases (EUCs), essential interactions (EI), security essential use cases (SecEUC) and security essential interactions (SecEI). These patterns are used to capture the security requirements and to recommend the mandatory security controls as well as to generate security requirements prototype.

6 Evaluation

6.1 Tool Usability

We conducted a preliminary evaluation of our developed prototype to evaluate its usability. The participants of this study were 40 students, comprising 16 males and 24

females. The average age of the students was 22 years old. They were final year students from the Bachelor of Computer Science majoring in Software Engineering. They have sufficient understanding of requirements engineering and were familiar with the concept and methodology of essential use cases and security .To explore the functionality of the tool, the students were provided with a set of requirements on "online car rental registration". They were informed that they will be observed and they are free to say aloud their responses of the tool while completing the task. The purpose of the observation was to identify the problems and misconceptions faced by the participants when using the tool. Further, the say aloud evaluation of the tool provided us with the users' spontaneous responses and suggestions for improvement. After the completion of the task, students were requested to answer four questions related to the usability [5] which consists of the usefulness, ease of use, ease of learning and satisfaction of the tool based on a five-level Likert scale. Students' responses for these questions were analysed and the results of the survey are shown in Figure 6.

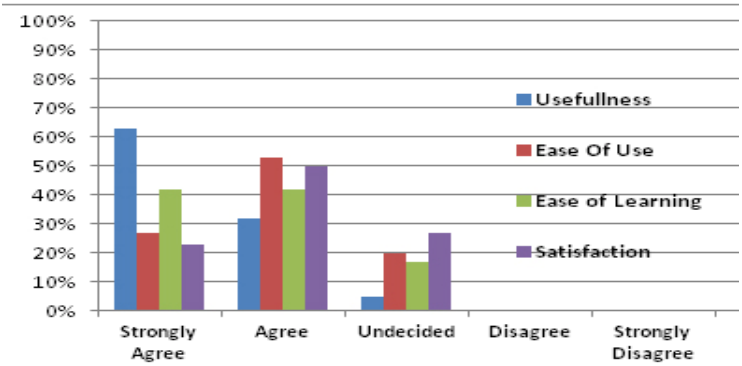

Fig. 6. Preliminary Usability Test Results

More than 90% of the participants agreed that the tool is useful, 70% agreed that the tool is easy to use, 80% agreed that the tool is easy to learn, and 70% were satisfied with the tool. None of the participants expressed disagreement to the four aspects evaluated in this survey. It can be concluded that our prototype tool is useful, easy to use, easy to understand and able to satisfy users. There are some enhancements that we need to consider for the refinement of our tool. Based on the say aloud responses, the suggestions given by the participants were mostly related to the improvement of the interface design of the tool and the provision of a user manual or a tutorial for users. Therefore, we plan to integrate the tool with existing security tools with the security features to ease the design and development phase. This stage is closely related to the work of the security engineers. Key threats to the validity of this findings are the selection bias due to our method of selecting the sample. The selection of participants for the preliminary study was based on one group of students. Hence, the subjects in the group were not homogenous with regards to the preference of the

interface design. However, they were similar in one or more of the subject-related variables, such as the agreement towards the easiness of using the tool.

6.2 Preliminary SecEUC Patterns Evaluation

We also conducted an evaluation of our SecEUC patterns using five experts in security requirements from IBM Corporation from India, Austria, France and Malaysia. They were invited to evaluate the correctness of our new SecEUC patterns library by evaluating the classification of security controls with the associated SecEUC and SecEI. They were given five sets of requirements from a small size security application requirements document. This consisted of 10 SecEUC, 50 SecEI and five basic security controls: confidentiality, integrity, availability, authentication and authorization. They were asked to answer a few questions using likert-scale and some open ended questions. The results of this expert evaluation are shown in Figure 7.

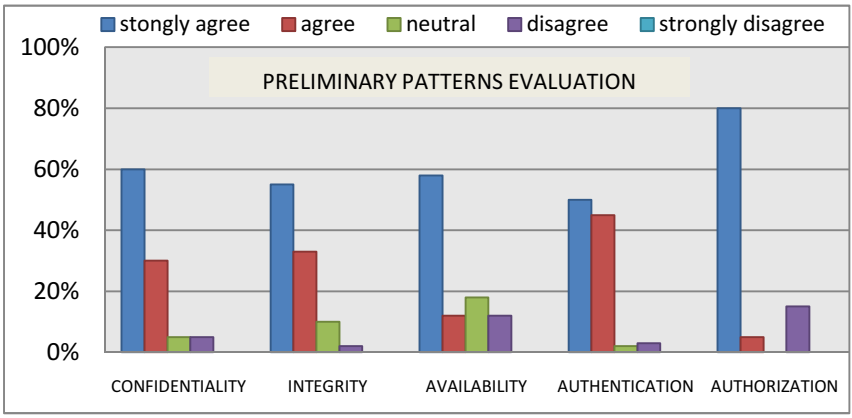

Fig. 7. Preliminary Patterns Evaluation Results

As shown in Figure 7, all of the general security controls rating have some disagreement. Based on this feedback, the disagreement does not mean that we have provided incorrect selection, but they would prefer some other security control classifications for each of the SecEUC. They requested more options for security controls for more complex requirements. For example, the security requirements "login" are currently mapped to the security controls "Authorization". Some experts think it should be mapped to another security control such as Authentication. They also offered some recommendations for correct and relevant security controls associated to a particular SecEUC. Here, we will update the security patterns from time to time in accordance with this feedback. Based on this study, the dependency on basic security services is not purely relevant for capturing more complex security requirements. Therefore, a further study on the patterns relates to more establish security standards is required.

7 Related Work

Many methods, approaches, techniques and tools have been used to capture security requirements. Viega [22], showed how to build security requirements in a structured manner that is conducive to iterative refinement. If this structured procedure is followed correctly according to the metrics for evaluation, it would serve as a framework that provides a significant improvement for the traditional methods that do not consider security at all. They also provided an example using a simple three-tiered architecture. The basic idea behind the way that CLASP handles security requirements is the performance of a structured walkthrough of resources, determining how they address each core security service throughout the lifetime of that resource. While it is obviously far more effective than any ad-hoc treatment of security requirements, this methodology is still new and immature.

Hussein and Zulkernine [23], proposed a framework for developing components with intrusion detection capabilities. The first stage of this framework is requirements elicitation, in which developers identify services and intrusions. In this framework, they capture users requirements regarding the services and functionalities provided by the components, and identify the unwanted or illegal usage of components by intruders. Intrusion scenarios are elicited through the use of misuse-cases of a UML profile called UMLintr. Yet, their proposed framework still needs an extension scope on UMLintr to other security requirements. UMLintr can be extended by exploring how to specify and handle other security requirements like authentication, authorization, integrity, etc. Their framework is also considered as complex intrusion scenarios. While Agile Security Requirements Engineering proposes the extension of agile practices to deal with security in an informal, communicative and assurance-driven spirit, it has its own limitation. It is only partially support consistency and does not support correctness and validation checking between security and business requirements.

i* frame is a modeling and analysis framework for organizational environments and their software systems, and is based on the intentionality relations between agents. Tropos [26] adopts the i* modeling framework and is an agent- oriented software system development methodology. However, it focuses on describing both organizational environment of a system and a system itself. A secure Tropos framework to model and analyze the security requirements is then built using the Secure Tropos methodology [27]. It especially addresses the problem of modeling security requirements through ownership, permission and delegation among the actors or agents involved in the software system. In our previous research [26] we reviewed a few related tools, such as STS-Tool [27], SecTro [28], SecReq [31], SREPPLine [30] and ST-Tool [31]. Many researchers have done great works on their research of security requirements engineering, particularly involving a tool that supports the security requirement engineering. We found that most work used a modeling approach, such as use cases, misuse cases, and UMLsec, to handle security requirements. UML models are the most commonly used [34], especially use case diagrams that are widely used by developers and requirements engineers to elicit and capture requirements. Kamalrudin et al. [7], [34] have shown that EUCs are useful to capture and validate the quality of requirements. EUCs also benefit the development process as they fit a

problem-oriented rather than solution–oriented approach, thus they have the potential to allow designers and implementers of the user interface to explore more possibilities. This approach allows for a more rapid development, whereby when using EUCs, it is no longer necessary to design an actual user interface. Although EUCs simplify captured requirements as compared to the conventional UML use cases, and it is beneficial to integrate the requirements and design [35][39], they have not explored the process of capturing security requirements.

8 Conclusion and Future Work

We have described a new approach to supporting security requirements capture and analysis using Essential Use Case models. Our prototype tool, SecMereq, provides support for extracting EUCs and security requirements from natural language text, prototype generation including security interfaces, and validation of extracted security requirements using a library of security related patterns. We have evaluated our tool in terms of both its usability for target end users, and for quality of the encoded EUC-based security patterns.

In future, we plan to enhance our security pattern library using a-well-established standard: Common Criteria. Then we intend to compare the efficacy of our tool with manual approaches of extracting security requirements. Additionally, we will work on the possibility of automating currently complicated usage of the standard to produce a simpler practice by using our tool support. This would then create the possibility of allowing a complete and consistent capture of security requirements from normal business requirements.

Acknowledgments. The authors also would like to acknowledge Universiti Teknikal Malaysia Melaka and the Ministry of Education Malaysia of the scholarship Mybrain15. We also would like to thank the funding of this ERGS research grant: ERGS/2013/FTMK/ICT01/UTEM/E00026 for funding this research.

References

1. Alam, M.: Software Security Requirements Checklist. International Journal of Software Engineering, IJSE 3(1), 53–62 (2010)
2. McGraw, G.: Building Security. In: Software Security. IEEE Security and Privacy, pp. 80–83 (2004)
3. Schneider, K., Knauss, E., Houmb, S., Islam, S., Jürjens, J.: Enhancing security requirements engineering by organizational learning. Requirements Engineering 17(1), 35–56 (2011)
4. Paja, E., Dalpiaz, F., Poggianella, M., Roberti, P., Giorgini, P.: STS-tool: Socio-technical Security Requirements through social commitments. In: Conference on IEEE International Requirements Engineering, pp. 331–332 (2012)
5. Kamalrudin, M., Hosking, J., Grundy, J.: Improving requirements quality using essential use case interaction patterns. In: Proceeding of the 33rd International Conference on Software Engineering - ICSE 2011, p. 531 (2011)

6. Elahi, G., Yu, E.: A Semi-automated Decision Support Tool for Requirements Trade-Off Analysis. In: IEEE 35th Annual Computer Software and Applications Conference, pp. 466–475 (2011)
7. Kamalrudin, M., Grundy, J., Hosking, J.: Tool Support for Essential Use Cases to Better Capture Software Requirements, pp. 327–336 (2010)
8. Mellado., D., et al.: A systematic review of security requirements engineering. Computer Standards and Interfaces (2010)
9. Ding, W., Marchionini, G.: A Study on Video Browsing Strategies. Technical Report, University of Maryland (1997)
10. Fröhlich, B., Plate, J.: The cubic mouse: A new device for three-dimensional input. In: Proceedings of the SIGCHI (2000)
11. Firesmith, D.: Specifying reusable security requirements. Journal of Object Technology (2004)
12. Salini, P.: Survey and analysis on Security Requirements Engineering. Journal Computers and Electrical Electrical Engineering, http://linkinghub.elsevier.com/retrieve/pii/S0045790612001644 (accessed October 1, 2012)
13. Corporation, M.: Simplified Implementation of the SDL. pp. 1–17 (2010)
14. Wiegers, K.E.: Software Requirements. O'Reilly (2009)
15. Souag, A., Salinesi, C., Comyn-Wattiau, I.: Ontologies for Security Requirements: A Literature Survey and Classification. In: Bajec, M., Eder, J. (eds.) CAiSE Workshops 2012. LNBIP, vol. 112, pp. 61–69. Springer, Heidelberg (2012)
16. Rodríguez, A., Fernández-Medina, E., Piattini, M.: Towards a UML 2.0 extension for the modeling of security requirements in business processes. In: Fischer-Hübner, S., Furnell, S., Lambrinoudakis, C. (eds.) TrustBus 2006. LNCS, vol. 4083, pp. 51–61. Springer, Heidelberg (2006)
17. Backes, M., Pfitzmann, B., Waidner, M.: Security in Business Process Engineering. In: van der Aalst, W.M.P., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 168–183. Springer, Heidelberg (2003)
18. Herrmann, G., et al.: Viewing Business Process Security from Different Perspectives. In: 11th International Bled Electronic Commerce Conference, Slovenia, pp. 89–103 (1998)
19. The SANS Institute, Determining the Role of the IA / Security Engineer, InfoSec Reading Room (2010)
20. Kamalrudin, M.: Automated Support for Consistency Management and Validation of Requirements". PhD thesis. The University of Auckland (2011)
21. Myagmar.: Threat Modeling as a Basis for Security Requirements. In: Proceedings of the ACM Workshop on Storage Security and Survivability, pp. 94–102 (2005)
22. Viega, J.: Building Security Requirements with CLASP. In: Proceedings of the Workshop on Software Engineering for Secure Systems Building Trustworthy Applications, SESS 2005, pp. 1–7 (2010)
23. Hussein, M., Zulkernine, M.: Intrusion detection aware component-based systems: A specification-based framework. Journal of Systems and Software 80(5), 700–710 (2007)
24. Du, J., et al.: An Analysis for Understanding Software Security Requirement Methodologies. In: Third IEEE International Conference on Secure Software Integration and Reliability Improvement, pp. 141–149 (2009)
25. Giorgini, P., et al.: Modeling security requirements through ownership, permission and delegation. In: 13th IEEE International Conference on Requirements Engineering (RE 2005), pp. 167–176 (2005)

26. Yahya, S., Kamalrudin, M., Sidek, S.: A Review on Tool Supports for Security Requirements Engineering. In: IEEE Conference on Open Systems, Sarawak, Malaysia (2013)
27. Paja, E., et al.: STS-tool: Socio-technical Security Requirements through social commitments. In: 2012 20th IEEE International Requirements Engineering Conference (RE), pp. 331–332. IEEE (2012)
28. Pavlidis, M., Islam, S.: SecTro: A CASE Tool for Modelling Security in Requirements Engineering using Secure Tropos. In: Proceedings of the CAiSE forum, CAiSE 2011, pp. 89–96 (2011)
29. Houmb, S.H., Islam, S., Knauss, E., Jürjens, J., Schneider, K.: Eliciting security requirements and tracing them to design: An integration of Common Criteria, heuristics, and UMLsec. Requirements Engineering 15(1), 63–93 (2010)
30. Mellado, D., Fernández-medina, E., Piattini, M.: Security Requirements Engineering Process for Software Product Lines: A Case Study and Technologies SREPPLine. pp. 1–6 (2008)
31. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: ST-Tool: A CASE tool for security requirements engineering. In: Proceedings of 13th IEEE International Conference on Requirements Engineering, pp. 451–452 (2005)
32. Kamalrudin, M., Hosking, J.G., Grundy, J.C.: Improving Requirements Quality using Essential Use Case Interaction Patterns. In: ICSE 2011, Honolulu, Hawaii, USA (2011)
33. Kaindl, H., Constantine, L., Pastor, O., Sutcliffe, A., Zowghi, D.: How to Combine Requirements Engineering and Interaction Design? In: 16th IEEE International Requirements Engineering, RE 2008, Barcelona, Catalunya, Spain, pp. 299–301 (2008)
34. Kamalrudin, M., Grundy, J., Hosking, J.: Managing Consistency between Textual Requirements. Abstract Interactions and Essential Use Cases, 327–336 (2010)
35. Yahya, S., Kamalrudin, M., Sidek, S.: The Use of Essential Use Cases (EUCs) to enhance the Process of Capturing Security Requirements for Accurate Secure Software. In: Proceeding of Software Engineering Postgraduates Workshop, SEPoW (2013)
36. Kamalrudin, M.: Automated Software Tool Support for Checking the Inconsistency of Requirements. In: 24th IEEE/ACM International Conference on Automated Software Engineering, ASE 2009. IEEE (2009)
37. Constantine, L.L., Lockwood, A.D.L.: Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design. ACM Press/Addison Wesley Longman, Inc. (1999)
38. Develop functional security requirements in Document security-relevant requirements retrieve, `https://www.owasp.org/index.php/Document_security-relevant_requirements` (accessed July 15, 2013)
39. Blackwell, A.F., et al.: Cognitive Dimensions of Notations: Design Tools for Cognitive Technology. In: Beynon, M., Nehaniv, C.L., Dautenhahn, K. (eds.) CT 2001. LNCS (LNAI), vol. 2117, pp. 325–341. Springer, Heidelberg (2001)
40. What is the Common Criteria (CC) in Common Criteria and Mutual Recognition retrieve from, `http://www.cybersecurity.my/myc` (accessed August 5, 2013)
41. Biddle, R., Noble, J., Tempero, E.: Essential use cases and responsibility in object-oriented development. In: Proceeding of the Twenty-Fifth Australasian Conference on Computer Science, Melbourne, Victoria, Australia, pp. 7–16. ACM (2002)
42. Biddle, R., Noble, J., Tempero, E.: Patterns for Essential Use Case Bodies. In: Proceedings of the 2002 Conference on Pattern languages of programs, CRPIT 2002, vol. 13, pp. 85–98. Computer Society, Australian (2002)