A Tagging Approach to Extract Security Requirements in Non-Traditional Software Development Processes

Annette Tetmeyer, Department of Electrical Engineering and Computer Science, University of Kansas, Lawrence, KS, USA

Daniel Hein, Automotive OEM, Garmin International, Olathe, KS, USA

Hossein Saiedian, Department of Electrical Engineering and Computer Science, University of Kansas, Lawrence, KS, USA

ABSTRACT

While software security has become an expectation, stakeholders often have difficulty expressing such expectations. Elaborate (and expensive) frameworks to identify, analyze, validate and incorporate security requirements for large software systems (and organizations) have been proposed, however, small organizations working within short development lifecycles and minimal resources cannot justify such frameworks and often need a light and practical approach to security requirements engineering that can be easily integrated into their existing development processes. This work presents an approach for eliciting, analyzing, prioritizing and developing security requirements which can be integrated into existing software development lifecycles for small organizations. The approach is based on identifying candidate security goals using part of speech (POS) tagging, categorizing security goals based on canonical security definitions, and understanding the stakeholder goals to develop preliminary security requirements and to prioritize them. It uses a case study to validate the feasibility and effectiveness of the proposed approach.

Keywords: *Application Development, Part Of Speech (POS) Tagging, Requirement Engineering Process, Security Requirements, Software Security*

1. INTRODUCTION

Software security is a complex, evolving problem that can be significantly improved by integrating security requirements into the early stages of software development rather than correcting security flaws after release (Allen, Barnum, Ellison, McGraw, & Mead, 2008). However, traditional software development life cycle (SDLC) processes tend to focus attention on functional requirements leaving non-functional requirements, such as security, as an aside or afterthought. This results in security requirements that are added later (McGraw, 2005) in the development cycle or

DOI: 10.4018/ijsse.2014100102

worse, after the product has been released in response to security events, market response, or regulatory demands.

There are several reasons for the reactionary response to software security. First, software engineers have a difficulty in building secure software due to a lack of software security awareness, training and education (Viega, 2005). Additionally, decisions about security may simply have been made based on the technology and capabilities available at the time the system was developed (i.e., early infrastructure systems). Finally, project cost constraints may focus resources on delivering functional requirements over non-functional requirements, such as security. Regardless of the reasons for this reactionary response to security, both software engineers and business stakeholders are becoming increasingly aware of software security needs.

Recent news reports of highly publicized data breaches have increased general awareness of the need to integrate security into software during development. In response, legislation at the state and federal level has also been increasing as the need for privacy and security becomes apparent. Such increased legislation is evidenced by the fact that nearly all states have enacted either security[1], or data breach[2] notification legislation. In general, software engineers have reacted to increased public awareness and legislative pressures by adding security mechanisms to existing systems on an ad hoc basis to mitigate risk. While subsequent mechanism based mitigation is a useful (and sometimes necessary) approach when addressing new, evolving, or previously unknown security risks, the approach often results in isolated, add-on countermeasures that are not cohesively integrated into the resulting system and its design. Money and time may be lost as software engineers work to prevent additional bugs as they retrofit existing systems with added security mechanisms.

However, if the security risk existed prior to development, then a security requirement could have addressed the risk by building security into the system from the start. In this case, implementing security mechanisms does not address the core problem that security requirements need to be integrated into software from the start, not mitigated later. The emerging field of security requirements engineering (SRE) seeks to address the special needs of integrating security requirements into the software development process.

1.1. Security Requirements Engineering

When it comes to specifying security requirements, stakeholders have different expectations. Integrating software security into development requires eliciting security requirements along with other requirements. From the business stakeholder viewpoint, software requirements are an extension of business goals that stakeholders wish to implement in a software product. Business goals generally represent desired functionality, but may also imply general security needs based on limited security knowledge that business stakeholders possess. Using these goals, the software requirements engineer must elicit functional and non-functional requirements. However, a hurdle to eliciting security requirements is the difficulty that stakeholders and software engineers have with explicitly expressing security needs. Security may again be expected, but verbalizing specific security requirements may be difficult. The combination of vague security goals and limited security specific resources often results in a requirements specification which may not include security specific requirements.

To remedy the lack of software security, requirements engineering approaches have been proposed to aid in the development of security requirements. Security requirements should be elicited and developed along with functional requirements and should be included as part of the software requirements specification. Best practices, enumerations, methodologies (Hallberg & Hallberg, 2006; Romero-Mariona, 2009; Mead, Hough, & Stehney, 2005), models (Luckey, Baumann, Méndez, & Wagner, 2010; McGraw, Migues, & West, 2012) and elicitation techniques (Ingoldsby, 2009) have been

proposed that are intended to better integrate security requirements into early phases of development. Each of these approaches may be useful when developing security requirements, but may be primarily appropriate for large organizations. The Security Quality Requirements Engineering (SQUARE) Methodology (Mead, Hough, & Stehney, 2005) is comprehensive and regimented. However, the sheer volume of pages in the SQUARE overview may be more documentation and rigor than is generally accepted by agile practitioners in small organizations. The Usage-Centric Security Requirements Engineering (USeR) Method (Hallberg & Hallberg, 2006) extracts security requirements using Voice of Customer (VOC) quality techniques, but only provides minimal direction on developing requirements, and then shifts attention to security mechanisms. Another method rooted in quality requirements is the Extended Activity-Based Quality Model (eABQM; Luckey, Baumann, Méndez, & Wagner, 2010). A key concept of eABQM is that well defined requirements in conjunction with a security requirements repository will promote requirements reuse. As a type of quality requirements, the eABQM models security requirements. This approach does not provide specifics for security requirements elicitation from existing requirements, but instead focuses on developing security requirements based on project parameters. Organizations well versed in security terminology, or with existing security modeling parameters, are more likely to benefit from this approach than a small organization seeking a method for quick and efficient security requirements elicitation. Secure and Usable Requirements Engineering (SURE; Romero-Mariona, 2009) aims to aid developers who may be lacking in security training. The process adopts activites from misuse cases and CLASP[3]-- comprehensive lightweight application security process, and splits its focus between two stages: security requirements and security testing. The security requirements stage models steps to evolve security statements to security needs to security requirements, but does not include specific details on how to achieve these activities. USeR and SURE have appealing aspects in that they evolve security requirements by extracting security needs from general requirements statements. The general concept of extracting requirements based on implied security statements should improve security requirements engineering and is desirable in a new approach. The Building Secuirty In Maturity Model (BSIMM), developed by McGraw, Chess, Migues, and West, gives overall guidance in improving security initiatives for an organization via activities carried out as part of various domains within BSIMM's Software Security Framework (SSF). Activities relevant to requirements (in SSF's intelligence domain), include using standards and attack models to supplement requirements elicitation. However, these recommended activities stand alone without specific guidance on their detailed execution (i.e., they are not prescriptive).[4] McGraw describes BSIMM as a "descriptive model" rather than providing "prescriptive guidance" (McGraw, Migues, & West, 2012). Therefore, BSIMM does not provide enough detail to be used primarily for security requirements elicitation and development.

The drawback to most of these approaches is that they require security expertise and resources that may be impractical to deploy in small, agile organizations. We characterize these organizations as having fewer than 25 members on the software development team, working within software development lifecycles on the order of months, juggling multiple projects at one time, and using or moving towards the use of agile methods (i.e., Scrum). Small, highly competitive software producers may see themselves as embracing agile development without specifically following a prescribed agile methodology (Ramesh, Cao, & Baskerville, 2010). These organizations may define themselves as agile because they see themselves as flexible and lean as well as embracing key tenets of agile philosophy. Close interaction of stakeholders, the ability to respond to evolving requirements and balancing documentation with delivering the product are key characteristics of these organizations. Regardless of whether

an organization is following a specific agile methodology, the impact on the process of requirements engineering is different for agile leaning organizations than those following traditional development methods.

1.2. Agile Requirements Engineering

Agile requirements engineering is different than traditional requirements engineering. Traditional requirements engineering includes sequential activities for eliciting, modeling, defining, and validating requirements that culminate in written documentation such as a software requirements specification. The requirements process is completed early in software development with the intent that only minor modifications are made after later stages of development commence. In contrast, agile requirements engineering processes are iterative and integrated throughout the software development lifecycle.

In an agile process, requirements are expected to evolve continually. While agile requirements engineering activities include the same activities as traditional approaches (Paetscsh, Eberlein, & Maurer, 2003), these activities do not have the same heavy up-front focus. Researchers have identified the following practices associated with agile requirements engineering (Ramesh, Cao, & Baskerville, 2010; Cao & Ramesh, 2008):

- Face-to-face communication over written specifications
- Iterative requirements engineering (requirements emerge throughout development)
- Manage requirements change through constant planning
- Extreme requirements prioritization (i.e., continual prioritization)
- Prototyping
- Review meetings and tests

Of these practices, continual prioritization, face-to-face communication and iterative requirements engineering are most relevant to capturing the evolving requirements that are characteristic of agile processes.

Ramesh et al further characterizes minimal documentation as one of the main challenges to iterative requirements engineering (Cao & Ramesh, 2008).

Traditional requirements engineering can be document heavy and include the creation of a formal software requirements specification. Agile requirements engineering does not mean that no written requirements are produced, but instead that the type of documentation varies by organization. For example, "agile leaning" organizations may use a request for proposal (RFP) or feature list that functions as a preliminary requirements artifact. Requirements elicitation is undertaken using these preliminary artifacts, just as it is in traditional processes, and may include traditional elicitation techniques as well as techniques associated with agile development (such as storyboarding). However, agile elicitation processes are not exhaustive and the expectation is that requirements will continue to evolve iteratively as the software is developed. Since agile organizations place less emphasis on "perfecting" requirements, the preliminary requirements artifact may be used to draft software requirements in a non-traditional form such as user stories. Regardless of the type of requirements documentation produced, there is still some form of written requirements that are used for customer approval as well as ongoing development. The main consideration is to focus on producing requirements in a workable format rather than spending precious time writing documentation.

An area of overlap in traditional and agile development methodologies is the emphasis on functional requirements and a lack of focus on non-functional requirements such as security. Agile focuses on quickly implementing functional requirements, not non-functional requirements. Approaches to handling non-functional requirements in agile development are "ill-defined" and approaches "need to include more explicitly the handling of non-functional requirements" (Paetscsh, Eberlein, & Maurer, 2003). Clearly, to reduce later rework,

requirements should be developed as early as possible during development. The overhead of spending large amounts of development time up-front to produce requirements can be significant if requirements are missed at the start. Security requirements may be initially overlooked since they are viewed as non-functional requirements and may not be well understood by all stakeholders. A certain amount of expertise and training is required to begin to understand security.[5] These organizations need a security requirements engineering approach that has an appropriate scope and does not require expert security resources. The approach must also be easy to incorporate into the regular requirements specification activities by specifically addressing requirements elicitation.

2. SECURITY REQUIREMENTS ELICITATION

We propose a security requirements elicitation approach that is part of the requirements elicitation phase of the software development lifecycle. The approach can be used for traditional software development processes by any size of development team, but is particularly applicable for small organizations following an agile development philosophy that is fast paced and iterative in nature. The approach is not targeted to a specific agile development method, such as Scrum, but can be followed during requirements elicitation activities for any development method. Small organizations with less than 25 personnel on the development team (business analysts, requirements engineers, developers, testers, project managers, quality assurance, etc.) and who are in the early stages of building a software security initiative are the ideal target organization. The activities in the approach are designed to take advantage of existing resources and artifacts to improve the elicitation of security requirements without imposing significant time or resource burden on the organization. The approach can be repeated, and previous results enhanced, to take advantage of the iterative nature of a typical agile development method.

Preliminary functional requirements artifacts are used as inputs to the process and can be used in many forms. Artifacts should be in text based electronic file formats in order for automated scanning to take place. Typical requirements artifacts are formal or informal software requirements specifications (SRS), user stories, use cases, RFPs, or other business documents outlining software functional requirements. These documents will be referred to as preliminary requirements artifacts throughout the approach description. After undertaking the activities in the approach, draft security requirements are output. The security requirements elicitation approach activities are defined as follows:

- Identify candidate security goals
- Categorize security goals based on security principles
- Understand stakeholder goals and develop preliminary security requirements
- Prioritize preliminary security requirements

Each activity defines inputs, roles, techniques, and output (see Figure 1). Inputs are requirements related artifacts. Roles are the development team and business stakeholders responsible for each activity. Techniques are applied to accomplish each activity and to develop security requirements. The final output for the approach is a prioritized security requirements artifact. Figure 2 illustrates the activities of the security requirements elicitation approach. The four key activities are sequential with iterations within each activity as needed. The output from each activity becomes the input for the subsequent activity. While overlap may occur between activities, each activity was chosen to signify a progression from unidentified or implied security requirements to defined and prioritized security requirements. The following sections will demonstrate the activities in the approach.

2.1. Identify Candidate Security Goals

Identifying security requirements can be difficult if stakeholders have difficulty expressing security related needs. Business stakeholders may imply the need to protect assets based on the limited knowledge of vulnerabilities and threats which leads to ambiguity when expressing security needs. The result may be requirements written with security terminology that implies security requirements but that are not explicitly defined. Development team stakeholders, such as the requirements engineer, developers, and testers, may also have difficulty extracting implied security requirements if they are not security experts. A small development organization is likely to have limited security experts dedicated to development and needs to efficiently and effectively work with business stakeholders to extract security requirements. If security terminology can be captured, candidate security goals can be identified that with further analysis can be used to develop security requirements. The identify candidate security goals activity is intended to discover implied security requirements based on the use of security terminology embedded within preliminary requirements.

A method that can be used to extract meaning from natural language is part of speech (POS) tagging. Online review opinion mining is an area of active research that uses POS tagging to extract sentiment or opinion from textual online reviews (Jindal & Liu, 2008; Mukherjee & Liu, 2012). The goals of online review opinion mining and extracting security requirements are similar. Natural language input contains meaning or sentiment that may not be easily inferred. Human experts and manual review methods are required to build a set of words or phrases that are meaningful based

Figure 1. Input, roles, techniques and output

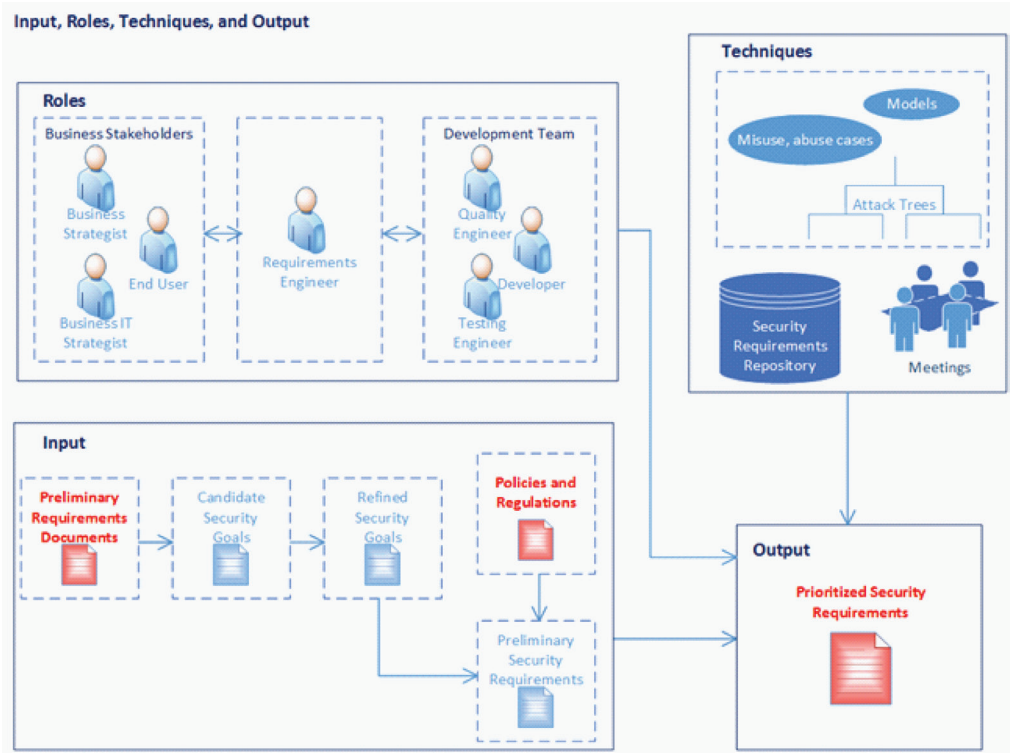
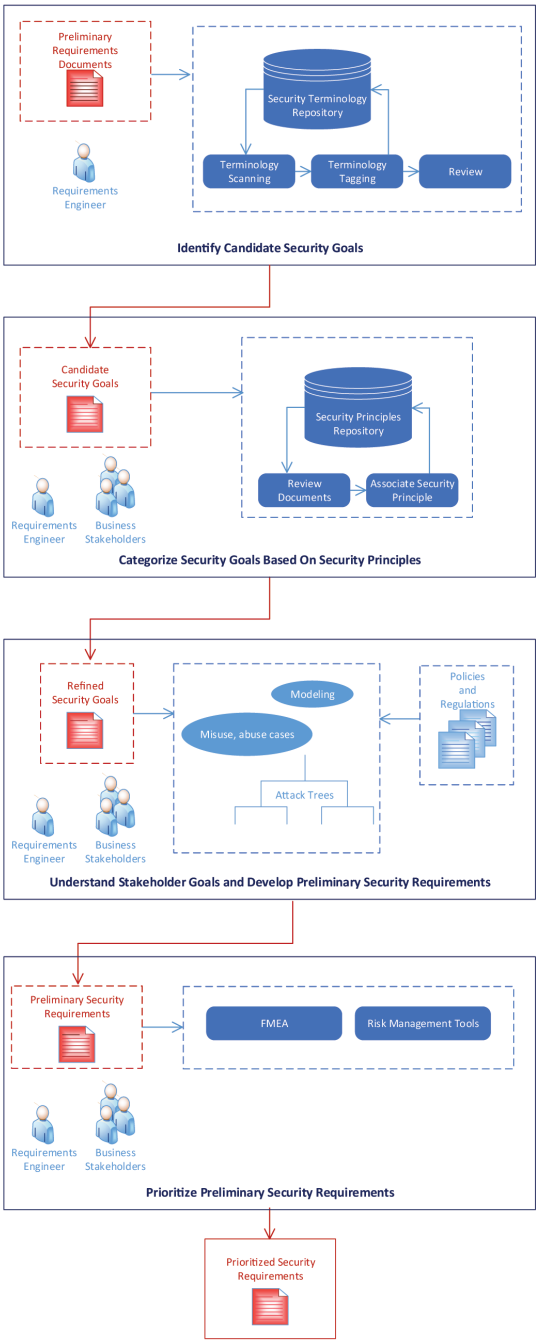


Figure 2. Security requirements elicitation approach



on the desired end result. Word frequency and proximity of terms analysis are common techniques used in opinion mining that may also be useful to security requirements elicitation. If the term "password" is frequently used, there may be an underlying security requirement related to the use of the term. If the terms "security" and "encryption" are located within close proximity of each other, then the terms may be associated with each other and could reveal an underlying security requirement. In order to capture implied security requirements, security terms within preliminary requirements artifacts can be tagged and follow-up analysis performed to determine if security requirements can be developed.

For the identify activity, the requirements engineer takes as input preliminary requirements artifacts for tagging. These artifacts can be draft software requirements specifications (SRS), request for proposals (RFPs), or other requirements specification artifacts that will be used to generate the final software requirements specification. Artifacts are scanned for commonly used security terminology. Generating commonly used security terms can be left up to the knowledge of the requirements engineer or a dictionary of security terminology can be used if available. Discovered security terminology and the location within the requirements artifacts are tagged for additional review. After all artifacts have been tagged, the requirements engineer reviews the requirements artifacts and identifies candidate security goals. Scanning and tagging of artifacts can be manually conducted or an automated tool can be utilized.

Candidate security goals (CSG) are general requirements written with implied security needs that may be developed into security requirements. For example, a functional requirement (FR) containing the term "password" could be written as in Box 1.

The safety of passwords seems like a reasonable requirement, but is still vague and does not reveal the underlying concern of the business stakeholders. Further examination of the requirement might yield information related to using passwords to protect the system from unauthorized access to information by using encryption mechanisms. The requirements engineer would generate a CSG such as in Box 2.

The identify activity continues until all candidate security goals are identified and output as an artifact for the categorize activity. Although it may be tempting to jump to further analysis of the identified candidate security goals at this

Table 1. FMEA Analysis of security requirements

Failure	Effect	Severity	Occurrence	Detection	RPN
unauthorized access	data viewed	3	7	9	189
unauthorized access	data stolen	9	4	9	324
unauthorized access	data corrupted	5	4	4	80
Standard Impact and Rating Scale for Severity, Occurrence or Detection					
Severity: 1 (insignificant) - 10 (catastrophic) Occurrence: 1 (extremely unlikely) - 10 (inevitable) Detection: 1 (absolutely certain to detect) - 10 (certain not to detect)					

Box 1. Functional requirement containing the term "Password"

FR–1: Passwords will be encrypted to ensure password safety.

Table 2. Security terminology statistics

	Artifacts Containing the Security Term		
Security Term	**Number of Artifacts**	**Percentage of Artifacts**	**Frequency of Security Term**
security	38	88.4%	551
access	34	79.1%	416
password	20	46.5%	237
authentication	15	34.9%	30
risk	15	34.9%	30
encryption	12	27.9%	20
authorized	10	23.3%	146
audit	7	16.3%	28
https	7	16.3%	14
permission	7	16.3%	86
privileges	6	14.0%	24
authenticate	5	11.6%	5
certificate	5	11.6%	205
encrypt	5	11.6%	12
certificates	4	9.3%	85
logon	4	9.3%	8
malicious	4	9.3%	8
deny	2	4.7%	3
authorize	0	0.0%	0
Total Artifacts	**43**		

Box 2. Candidate security goal

CSG-1: The system shall protect against unauthorized access by requiring encrypted user passwords.

stage, the role of the requirements engineer is to solely identify the candidate security goals at this stage. The categorize activity conducted next is designed to include additional stakeholders in a collaborative manner to analyze the candidate security goals.

2.2. Categorize Security Goals Based on Security Principle

Candidate security goals identified from the previous activity are used as input for the categorize activity. The requirements engineer can assess the goals for quick categorization to facilitate efficient communication and then work with business stakeholders to review

all tagged requirements artifacts. Business stakeholders should be educated on general security principles to improve the review and to improve the security education of all stakeholders. The key difference between the identify and categorize activities is that additional stakeholders are included in a collaborative environment for additional analysis. The requirements engineer leads the activity but also aids in educating business stakeholders on general security principles. During this activity, each security goal is categorized based on a security principle (SP) in order to facilitate additional stakeholder elicitation. Security principles provide a common language and understanding to improve stakeholder education and communication. Confidentiality, integrity, and availability principles (also referred to as CIA) are the key security principles, but other principles (e.g., non-repudiation) can be defined as well.[6] These three security principles may be defined as in Box 3.

If a CSG cannot be categorized, additional elicitation and analysis can be iteratively undertaken among stakeholders. If the CSG still cannot be categorized after additional iterations, it will fail the activity and the CSG will be discarded. CSGs are broad in definition at this point and will be further refined in subsequent activities.

2.3. Understand Stakeholder Goals and Develop Preliminary Security Requirements

Using the refined security goals from the categorize activity, the requirements engineer and business stakeholders seek to further understand the implications of the security goals. This allows the requirements engineer to further understand the business needs in order to develop preliminary security requirements. Additional artifacts such as policies and regulations are also used as input to this activity. The requirements engineer chooses techniques and tools to further elicit information from the business stakeholders. Face-to-face or virtual meetings are a good choice of techniques for generating discussion. The choice of tools is likely to be influenced by the requirements engineer but could include generating misuse or abuse cases, attack trees, or other security related modeling tools. Whereas the categorize activity may be conducted in one session, the understand activity may require multiple sessions with various stakeholders to complete. This allows different groups of stakeholders to work efficiently on specific tasks and collaborating as needed. The output from this activity is a set of preliminary security requirements based on the CSGs. Continuing with the previous example, the activity generates a preliminary security requirement (PSR) from CSG-1 such as in Box 4.

The development of PSRs demonstrates the further refinement from implied to well-defined security requirements. The output PSRs are used as input for the prioritize activity.

2.4. Prioritize Preliminary Security Requirements

Preliminary security requirements need to be prioritized to generate prioritized security requirements. Regardless of the type of requirements generated, lists of requirements tend to begin with a list of "wants" instead of "needs". Since it is not feasible to implement all requirements, the key goal of the prioritize activity is to whittle down the list of PSRs.

Box 3. Security principles

SP-1 Confidentiality: protect against unauthorized disclosure of information
SP-2 Integrity: protect against unauthorized modification or destruction of information
SP-3 Availability: protect against disruption of access to or use of **information of an information system**

Box 4. Preliminary security requirement from CSG-1

PSR-1: The system shall protect the confidentiality and integrity of data from unauthorized access by requiring encrypted user passwords.

During this activity, the requirements engineer continues to work with business stakeholders to analyze the input PSRs. Recommended analysis techniques are risk management tools commonly used by the stakeholders to foster familiarity and enhance communication. The choice of tool should be efficient and effective to meet the needs of a broad range of stakeholder backgrounds. We will demonstrate a technique not commonly used in software risk analysis, Failure Modes and Effects Analysis (FMEA), that can be quickly and intuitively applied to the task of prioritizing PSRs.

FMEA is an analysis and decision making tool often associated with quality and Six Sigma methodologies. A failure mode is the manner in which something might fail. Effects analysis is the study of the consequences of these failures. FMEA is used to identify, estimate, prioritize, and reduce the risk of failure. As a software engineering tool, FMEA is not widely used, but has advantages over other analysis tools in that it is easy to implement and can be used by a broad audience. A requirements engineer can use FMEA to elicit security related information from stakeholders, prioritize the data, and present an analysis of the risks associated. The prioritized risks allow for informed decision making to choose which actions to consider. This approach is very useful to communicate and clarify the impact of technical materials in an easy to understand format.

FMEA begins by determining the modes of failure and the effects of failure. Next, failures and effects are rated based on severity, occurrence and detection. A standard scale for severity, occurrence and detection can be adopted as a starting point for FMEA analysis but experienced FMEA users may wish to develop more refined ratings scales. A standard scale ranges from a low of 1 to a high of 10. The ASQ (formerly American Society for Quality) provides an overview of FMEA and recommends the following generic scales (ASQ, 2013):

- **Severity:** 1 is insignificant and 10 is catastrophic
- **Occurrence:** 1 is extremely unlikely and 10 is inevitable
- **Detection:** 1 is absolutely certain to detect and 10 is certain not to detect

Each rating between 1 and 10 should include a definition or criteria to differentiate between rankings. Standard FMEA scales are readily available on the internet or the organization can develop a custom set of scales. Finally, ratings are used to calculate a risk priority number (RPN). The RPN is calculated as the product of the risk ratings:

RPN = (severity ranking)(occurrence ranking)(detection ranking)

Continuing from the previous activities, Table 1 demonstrates analyzing the preliminary security requirement related to unauthorized access (PSR-1). Failure is unauthorized access and the effect of this failure could be data that is viewed, stolen or corrupted. The security requirements engineer could generate a preliminary table and follow-up with business stakeholder or all stakeholders could be involved at the start of analysis. Ratings for severity, occurrence and detection are determined by the stakeholders and the RPN is calculated. The resulting RPN generates a prioritized list of potential security requirements. In this scenario, the risk of data being stolen due to

unauthorized access significantly outweighs other effects. Stakeholders can either reject or accept each scenario from the FMEA analysis to determine if any of the RPNs are low enough to be rejected. Each failure and effect can also be split into separate security requirements and further refined. Once accepted or rejected, the requirements engineer and business stakeholders will refine the preliminary security requirements until a list of prioritized security requirements has been generated. Assuming that the RPN for this scenario is accepted and the PSR does not require splitting, the PSR is converted to a security requirement (SR) as shown in Box 5.

A sample template for the completed security requirements elicitation approach is shown in Figure 3.

3. POS SCANNING AND TAGGING TOOL

The main input for approach is a set of preliminary security requirements artifacts. During the identify candidate security goals activity, the requirements engineer must use POS scanning and tagging. For a small set of artifacts, the process can be accomplished manually. However, the use of an automated tool would improve efficiency, accuracy, and analysis capabilities. For this study, the authors chose to develop a software tool to scan and tag text based documents as well as to gather statistics on the processed artifacts. The first step was to choose a set of security terminology for scanning. Single terms (unigrams) were chosen, but the tool could accommodate short phrases. In addition, similar terms ("authorize" vs. "authorized") and plural terms ("password" vs. "passwords") can be accommodated by employing stemming methods.

Once the security terms are defined, the requirements artifact can be scanned and tagged which will gather the location and frequency of each term. A data set is output for each artifact. Tagging aids with subsequent activities for the approach by improving the navigation and analysis within each artifact. Identification of the location for a particular term or all terms within an area of the document is also available for ease of analysis. The POS scanning and tagging tool could be easily modified to include proximity of tagged terms, particularly for duplicate terms. Future versions of the tool could include this feature to improve the efficiency of the approach.

4. ANALYSIS AND FEASIBILITY OF THE APPROACH

Analysis and feasibility of the proposed solution should be taken into consideration given that we are targeting small organizations practicing agile development methods. Drawbacks to other approaches discussed in the background included approach complexity, resources required and security expertise. For this approach, preliminary requirements artifacts and a list of security terms for tagging are required to begin the activities. Minor training on security principles and FMEA analysis (or another risk analysis approach) is also required. The approach can mature with the security expertise of stakeholders. These features make the approach desirable for an organization undertaking a software security initiative with minimal upfront effort. The approach is flexible and additional techniques can be easily integrated if desired. Finally,

Box 5. Security requirement

SR-1: The system shall protect the confidentiality and integrity of data from unauthorized access by requiring encrypted user passwords.

Figure 3. Sample template for security requirements elicitation

Security Requirements Elicitation

Functional Requirement

FR - 1 Passwords will be encrypted to ensure password safety.

1 Identify candidate security goals

Candidate Security Goals (CSG)

CSG - 1 The system shall protect against unauthorized access by requiring encrypted user passwords.

2 Categorize security goals based on security principle

Security Principles

SP-1 Confidentiality: protect against unauthorized disclosure of information

SP-2 Integrity: protect against unauthorized modification or destruction of information

SP-3 Availability: protect against disruption of access to or use of information of an information system

Apply security principle(s) to CSG

CSG - 1 SP-1, SP-2

3 Understand stakeholder goals and develop preliminary security requirements

Preliminary Security Requirement (PSR)

PRS - 1 The system shall protect the confidentiality and integrity of data from unauthorized access by requiring encrypted user passwords.

4 Prioritize preliminary security requirements

PSR	Effect	FMEA RPN	Accept/Reject
PRS - 1	Data viewed	189	Accept
PRS - 1	Data stolen	324	Accept
PRS - 1	Data corrupted	80	Accept

Prioritized Security Requirements (SR)

SR - 1 The system shall protect the confidentiality and integrity of data from unauthorized access by requiring encrypted user passwords.

Notes

All of the candidate security requirements were previously identified as general security requirements. FMEA analysis confirms the need for the requirements. These requirements should be converted to security requirements.

the approach can be undertaken iteratively to integrate with the existing agile software development methods.

Tagging of terms can be easily automated if preliminary requirements artifacts are in text based files. Requirements developed using a software development tool (such as JIRA) may be exportable to a text file for automated scanning or a utility may be available to find security terms. In the case that requirements are not captured electronically (note cards, whiteboard), scanning and tagging of terms will have to be done visually. For a small-scale iterative project, the volume of requirements should be minimal and tagging can be accomplished with minor impact on time and personnel resources. To begin tagging, a list of security terms should be determined. The security terms can be determined by expert judgment or from a glossary of software security terms. For this approach, an initial list of terms was chosen using the author's judgment

based on experience and by reviewing security specific software requirements available to the authors (see Table 2). The list of security terms is not exhaustive and can be modified to meet the stakeholder's needs. If tagging is not automated, using a concise list of terms minimizes the resources needed as well. After an initial list of security terminology was chosen, a set of 43 sample requirements artifacts were tagged for the security terms. Scanning and tagging was accomplished using a software tool developed by the authors. The tool scanned text-based requirements artifacts using the security terms. A set of statistical data for each artifact was output and data was compiled as shown in Table 2. Terms such as "security" and "access" were found in a large percentage of artifacts (88.4% and 79.1% respectively) and had the highest frequency. "Password", "authentication", and "risk" were found in over 35% of the artifacts. The frequency of terms such as "security", "access" and "password" appears to indicate implied security requirements even though most artifacts lacked security specific requirements.

Once initial tagging is complete, the requirements engineer will examine the tagged artifact. It is likely that the number of tagged items to examine can be reduced due to proximity of terms or false positives. In the example used earlier, a functional requirement may have been tagged for the term "password".

"Password" is used twice in close proximity and is therefore somewhat redundant if the total number of tagged terms is considered (see Box 6). In this case, both of these tagged terms would be considered as one reducing the amount of effort needed to review the artifact. A false positive is a term that was tagged that is not associated with security goals. An example of a false positive might be associated with the term "certificate". There may be instances in which the use of "certificate" has no relation to security and could be eliminated from review. Reducing the number of terms for review due these reasons can be accomplished in a relatively short time (and could be automated in the future) and is feasible for a small organization.

The requirements engineer plays a key role in all activities and other stakeholders are involved in categorize, understand and prioritize activities. These subsequent elicitation activities require stakeholder meetings to develop security requirements, but do not require significant expertise or training. Conducting the FMEA analysis will take minor training and startup time to determine failure modes, expected effects and appropriate scales to be used to calculate the RPN. However, the process is easy to understand by non-technical stakeholders and guidance by the requirements engineer makes FMEA analysis a feasible technique. Additional models and techniques that are currently in use by the requirements engineer are not excluded and can also be included in the approach if desired. Therefore, the proposed security requirements elicitation approach is a feasible alternative to other security requirements elicitation approaches for small organizations following agile leaning development methods.

5. SUMMARY

This paper describes an approach for eliciting security requirements using security term tagging which can be implemented by small, agile organizations. Key elements of the elicitation approach are (1) identifying security goals, (2) categorizing goals by security principle, (3) understanding stakeholder goals to develop

Box 6. Functional requirement

FR–1: Passwords will be encrypted to ensure password safety.

preliminary requirements and (4) prioritizing security requirements for inclusion into the SRS document. Stakeholder roles, input artifacts, techniques and output artifacts are defined for each phase of the solution. The approach outlines a basic structure that can be easily implemented due to a flexible, iterative design and minimal upfront resources required. The approach should be incorporated into the software development process during requirements elicitation in order to reduce cost and rework at later stages of development.

A variety of preliminary requirements artifacts can be tagged for security terms using automated or manual methods. Tagging security terms jump starts the elicitation process and focuses efforts on specific areas of the requirements artifact for further examination. Review of tagged terms indicates that security terms are typically grouped in close proximity and duplicates can be identified and untagged. False positives, or security terms that are not associated with security goals, are also manually untagged. The resulting set of tagged security terms can then be analyzed using the approach. A key component of prioritization is the implementation of FMEA analysis which has roots in Six Sigma methodologies. FMEA analysis has not commonly considered as an analysis tool that can be used as part of the requirements elicitation, but has advantages in that it is easy to understand by non-technical stakeholders, is quick and aids in prioritization of security requirements. RPN results are based on rating risk based on frequency, occurrence and detection each of which can be addressed individually to reduce risk. The approach is flexible and the scope of effort can be adjusted to accommodate organizational resources for a software project.

Future work on tagging could improve the approach. Frequency of terms, proximity and associations between terms may be more significant than developing a large dataset of security terms. Expanding the terminology to include short phrases of related terms should also be explored to improve understanding of security goals. The relationship between a combination of terms and association with specific security principles should be explored.[7] Finally, failure modes and effects analysis could be used to automatically generate abuser stories which are commonly used with agile development elicitation and modeling techniques.

REFERENCES

Allen, J. H., Barnum, S., Ellison, R. J., McGraw, G., & Mead, N. R. (2008). *Software security engineering: a guide for project managers*. Addison-Wesley.

ASQ. (11/21/13). *Failure Mode Effects Analysis (FMEA)*. Available: http://asq.org/learn-about-quality/process-analysis-tools/overview/fmea.htmlhttp://asq.org/learn-about-quality/process-analysis-tools/overview/fmea.html

Cao, L., & Ramesh, B. (2008). Agile requirements engineering practices: An empirical study. *Software, IEEE*, *25*(1), 60–67. doi:10.1109/MS.2008.1

Hallberg, N., & Hallberg, J. (2006). The Usage-Centric Security Requirements Engineering (USeR) Method. In *Information Assurance Workshop (IEEE 2006)* (pp. 34-41). doi:10.1109/IAW.2006.1652074

Ingoldsby, T. R. (2009). *Attack tree-based threat risk analysis*. Retrieved from http://www.amenaza.com/downloads/docs/AttackTreeThreatRiskAnalysis.pdfhttp://www.amenaza.com/downloads/docs/AttackTreeThreatRiskAnalysis.pdf

Jindal, N., & Liu, B. (2008). Opinion spam and analysis. In *Proceedings of the international conference on Web search and web data mining*, Palo Alto, CA. doi:10.1145/1341531.1341560

Luckey, M., Baumann, A., Méndez, D., & Wagner, S. (2010). Reusing security requirements using an extended quality model. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, Cape Town, South Africa. doi:10.1145/1809100.1809101

G. McGraw, "The Security Lifecycle-The 7 Touchpoints of Secure Software-Just as you can't test quality into software, you can't bolt security features onto code and expect it to become hack-proof Security," *Software Development,* vol. 13, pp. 42-43.

McGraw, G., Migues, S., & West, J. (2012). *The Building Security In Maturity Model*. Available: http://www.bsimm.com/http://www.bsimm.com/

Mead, N. R., Hough, E., & Stehney, I. I. T. (2005). *Security Quality Requirements Engineering*. Retrieved from http://www.sei.cmu.edu/library/abstracts/reports/05tr009.cfmhttp://www.sei.cmu.edu/library/abstracts/reports/05tr009.cfm

Mukherjee, A., & Liu, B. (2012). "Modeling review comments," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, Jeju, Republic of Korea (pp. 320-329).

Paetsch, F., Eberlein, A., & Maurer, F. (2003). Requirements engineering and agile software development. In Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on (pp. 308-313).

Ramesh, B., Cao, L., & Baskerville, R. (2010). Agile requirements engineering practices and challenges: An empirical study. *Information Systems Journal*, *20*(5), 449–480. doi:10.1111/j.1365-2575.2007.00259.x

Romero-Mariona, J. (2009). "Secure and Usable Requirements Engineering," presented at the Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering. doi:10.1109/ASE.2009.81

Viega, J. (2005). Security - Problem Solved? *Queue*, *3*(5), 40–50. doi:10.1145/1071713.1071728 PMID:16467894

ENDNOTES

1 http://www.ncsl.org/issues-research/telecom/overview-security-breaches.aspx

2 http://datalossdb.org/us_states

3 https://www.owasp.org/index.php/Category:OWASP_CLASP_Project

4 BSIMM collects together common activities and practices from several organizations believed to be the best examples of firms with exceptional software security. Since there are 112 activities in BSIMM, organizations utilizing BSIMM are free to chose a subset of activities that they feel most relevant.

5 Our experience backs this claim. Many managers and software developers do not understand even some of the most well-known attacks (e.g., stack smashing buffer overflow) and do not understand the enabling factors that allow such attacks to succeed. Additionally, each development domain may have its own concerns and regulations. For example, banking is much different than embedded consumer electronics.

6 Non-repudiation may be related to keywords such as 'log' or 'trace', in order to provide an audit record. Non-repudiation was an essential part of an update system we developed. The high level security goal was to prevent insiders from abusing a tool that could be used to jailbreak (root) Android phones. To keep insiders honest, the system logged their network username when they accessed the tool. Their name was also used as a watermark inside the update used for jailbreaking.

7 A complementary semantic ontology model is available here: http://disi.unitn.it/~massacci/Publications/MASS-MYLO-ZANN-07-ONTOBOOK.pdf The above model might be able to provide additional context and keywords for specific roles and the delegation of trust. The primary advantage of this ontology is that it enables modeling non-technical controls and organizational policies along with those that are technical in nature. As such, it looks at the interactions between people and technical systems required for overall security.

Annette Tetmeyer is a Ph.D. candidate in computer science at the University of Kansas. Her research interests include security requirements engineering, data mining, human computer interaction, and engineering education. In addition to experience in private industry, she has taught a variety of undergraduate and graduate engineering courses at the University of Kansas. She received her MS in Computer Science from the University of Kansas (2013), her MS in Engineering Management from the University of Kansas (1998) and BS in Mechanical Engineering from Iowa State University (1993).

Daniel Hein is a senior software engineer at Garmin International where he works on handset and PDA software for Garminfone™, Nüvifone™, Garmin Mobile XT™, and iQue™ series products. Hein developed software for the iQue™ that controlled mixing and playback of music, voice recordings, and navigation streams. Hein holds a M.S. in Computer Engineering from the University of Kansas and a B.S. in Computer Engineering from Iowa State University, and is a member of IEEE. Hein is currently working on his Ph.D. in Computer Science with emphasis in Secure Software Engineering. Hein's research interests include security, software engineering, and machine learning.

Hossein Saiedian (Ph.D., Kansas State University, 1989) is currently an associate chair, director of IT degree programs, and a professor of computing and information technology at the Department of Electrical Engineering and Computer Science at the University of Kansas (KU) and a member of the KU Information and Telecommunication Technology Center (ITTC). Professor Saiedian has over 150 publications in a variety of topics in software engineering, computer science, information security, and information technology. His research in the past has been supported by the NSF as well as other national and regional foundations. Professor Saiedian has been awarded a number distinguished awards, including the KU's highly prestigious Kemper award for excellence in teaching, the University of Nebraska's awards in excellence in research and excellence in teaching, and was ranked among the top-10 software engineering scholars by the Journal of Systems and Software. At KU, he has graduated more than 65 MS and Ph.D. students. Professor Saiedian is credited with a number of degree programs he has developed, including the existing Master of Science in Information Technology (MSIT), and a new Bachelor of Science in Information Technology (BSIT) at the University of Kansas. Professor Saiedian is a member of the ACM, a senior member of the IEEE, and was among the first group to become IEEE's Certified Software Development Professional.