A POS Tagging Approach to Capture Security Requirements within an Agile Software Development Process

Annette Tetmeyer

Submitted to the graduate degree program in Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Committee Members:

_____
Dr. Hossein Saiedian
Professor and Thesis Adviser

_____
Dr. Arvin Agah
Professor

_____
Dr. Prasad Kulkarni
Associate Professor

Date Defended _____

The thesis committee for Annette Tetmeyer certifies that this is the approved version of the following thesis:

A POS Tagging Approach to Capture Security Requirements within an Agile Software Development Process

Committee Members:

_____
Dr. Hossein Saiedian
Professor and Thesis Adviser

_____
Dr. Arvin Agah
Professor

_____
Dr. Prasad Kulkarni
Associate Professor

Date Approved _____

Abstract

Software use is an inescapable reality. Computer systems are embedded into devices from the mundane to the complex and significantly impact daily life. Increased use expands the opportunity for malicious use which threatens security and privacy. Factors such as high profile data breaches, rising cost due to security incidents, competitive advantage and pending legislation are driving software developers to integrate security into software development rather than adding security after a product has been developed. Security requirements must be elicited, modeled, analyzed, documented and validated beginning at the initial phases of the software engineering process rather than being added at later stages. However, approaches to developing security requirements have been lacking which presents barriers to security requirements integration during the requirements phase of software development. In particular, software development organizations working within short development lifecycles (often characterized as agile lifecycle) and minimal resources need a light and practical approach to security requirements engineering that can be easily integrated into existing agile processes.

In this thesis, we present an approach for eliciting, analyzing, prioritizing and developing security requirements which can be integrated into existing software development lifecycles for small, agile organizations. The approach is based on identifying candidate security goals, categorizing security goals based on security principles, understanding the stakeholder goals to develop preliminary security requirements and prioritizing preliminary security requirements. The identification

activity consists of part of speech (POS) tagging of requirements related artifacts for security terminology to discover candidate security goals. The categorization activity applies a general security principle to candidate goals. Elicitation activities are undertaken to gain a deeper understanding of the security goals from stakeholders. Elicited goals are prioritized using risk management techniques and security requirements are developed from validated goals. Security goals may fail the validation activity, requiring further iterations of analysis, elicitation, and prioritization activities until stakeholders are satisfied with or have eliminated the security requirement. Finally, candidate security requirements are output which can be further modeled, defined and validated using other approaches. A security requirements repository is integrated into our proposed approach for future security requirements refinement and reuse. We validate the framework through an industrial case study with a small, agile software development organization.

Acknowledgements

I would like to thank the members of my committee, Dr. Arvin Agah and Dr. Prasad Kulkarni, for their time and assistance during this undertaking. I have had the opportunity to take classes with both of them and appreciate the time, expertise and enthusiasm that they contribute to students.

I would especially like to thank Dr. Hossein Saiedian for his support, instruction and mentoring over the past few years. His impressive knowledge, commitment to quality and exceptional work ethic drive those of us who have had the chance to work with him to improve ourselves. I don't know how he does it but am very grateful that I have had the opportunity to pursue my academic endeavors with his exceptional guidance.

Table of Contents

List of Tables

List of Figures

1 Introduction

Software security is a complex, evolving problem that has only recently begun to receive additional attention. One area that has previously received less attention is building security into software (Mead, Hough, & Stehney II, 2005) rather than correcting security flaws after release. Integrating security requirements into the software development life cycle (SDLC) from the start can significantly improve software security and reduce rework at later stages. However, traditional SDLC processes tend to focus attention on functional requirements leaving non-functional requirements, such as security, as an aside or afterthought. Current processes exist to aid the development of security requirements, but these processes have several drawbacks when security goals are vague or difficult for stakeholders to quantify. In particular, small software development teams have not only limited personnel resources, but may also be working within shorter time frames than large scale software development projects. The need to balance resources for fast paced software development projects and to remain competitive in the market has influenced the shift from traditional to agile development processes (Boehm, 2002). Therefore, there is a need for a security requirements approach to aid small, agile development organizations with the elicitation and development of security requirements when stakeholders have a difficulty explicitly expressing software security needs. The

approach must be easy to implement, efficient and reusable regardless of the development style followed by an organization.

1.1 **Problem Statement**

A hurdle to eliciting security requirements is the difficulty stakeholders and software engineers have in explicitly expressing security needs. Stakeholders involved with requirements development will have varying levels of awareness, education and training related to security. Business goals generally represent desired functionality, but may also imply general security needs. The software requirements engineer must be skilled in eliciting functional and non-functional requirements, but in small organizations, education and resources to develop requirements may be lacking. Small organizations may also be drawn to agile development processes due to the desire to produce software quickly while responding to customer needs (Peeters, 2005; Savolainen, Kuusela, & Vilavaara, 2010). For this work, we are focusing on small, agile organizations. These organizations need to balance resources effectively and are not likely to have devoted resources to expertly guide the development of security requirements. Therefore, there is a need to discover and extract implied security goals from existing requirements artifacts in order to develop security requirements. This thesis proposes a unique approach to capture security requirements within an agile software development process by utilizing part of speech (POS) tagging, analysis tools and a security requirements repository.

1.2 Significance

Security has predominately been an afterthought to the software development process. Functional requirements are developed at the beginning of the process, but non-functional requirements such as security are often overlooked. This results in security requirements that are "bolted on" (McGraw, 2005) later in the development cycle or worse, after the product has been released in response to security events, market response or regulatory demands. Adding requirements at later stages of development significantly impacts project cost. As security requirements become integrated, software product quality is expected to improve and rework due to security requirements added later in the process should decrease. However, existing security requirement approaches have drawbacks. Modeling tools such as misuse cases, abuse cases, and attack trees assume that security goals have already been identified and are used to refine security requirements. Methodologies may be useful when developing comprehensive security development best practices and policies, but do not focus specifically on security requirements. Other approaches specifically address the development of security requirements, but can be cumbersome and lengthy for agile development teams.

The reasons for the lack of attention to software security are many. Software engineers and stakeholders may lack general security awareness and education. Project constraints may focus resources on delivering functional requirements leaving non-functional requirements such as security a lower priority. In other cases, decisions about security may simply have been made based on the technology

capabilities at the time. Consider the development of supervisory control and data acquisition (SCADA) systems that manage power plants and public utilities. Early infrastructure systems were not networked and reachable by the outside world to the degree that they are today. Physical security was more important than system security and specific system security requirements were either limited or undefined. As awareness of system vulnerabilities increased, security requirements and mechanisms were added to existing systems on an ad hoc basis.

Software security vulnerability awareness increased not only for critical system software, but also for common software that impacted the general public. Highly publicized data breaches, such as the 2003 theft of over 45 million credit and debit card data from T.J. Maxx (Jewell, 2007), increased awareness among the general public. Legislation at the state and federal level has also been increasing as the need for privacy and security becomes apparent. Some legislation has been long-standing, such as the Privacy Act of 1974, but additional legislation has recently been enacted. Privacy and security rules for the Health Insurance Portability and Accountability Act (HIPAA) were enacted at the federal level in 2003[1]. Nearly all states have enacted either security[2] or data breach[3] notification legislation. Vulnerability awareness also drove increased security awareness among software engineers who frequently turned to implementing security mechanisms in order to

[1] http://www.hhs.gov/ocr/privacy/
[2] http://www.ncsl.org/issues-research/telecom/overview-security-breaches.aspx
[3] http://datalossdb.org/us_states

mitigate risk. However, this does not address the core problem that security requirements need to be built into software from the start, not addressed later.

Small organizations with fewer than twenty people on the development team are likely to operate with limited resources. A single person may be responsible for multiple roles, such as performing both quality engineer and tester roles. A single security engineer may be available, but it is unlikely that a security engineering team exists. For agile organizations, development will be iterative and extensive documentation will be less valuable than developing a working product[4]. Development schedules are likely to be shorter placing increased emphasis on project cost and time constraints. Therefore, integrating security requirements into the software development process for small, agile organizations requires careful balancing of project resources and constraints.

Increasing security threats, lack of software engineering security skills, consumer expectations for secure software and project constraints for small, agile organizations demonstrates the need to improve security requirements engineering. The increased complexity and integration of systems increases attack surfaces and makes it difficult to understand software vulnerabilities. Software engineers traditionally do not receive adequate training or attention to security to address software vulnerabilities. Publicity of the latest data breach or widespread virus now makes front page news. In addition, introducing project requirements strain limited project resources in terms of cost, time and personnel. Traditional software

[4] http://agilemanifesto.org/

development processes have focused on cost and time constraints which leave little room for additional requirements development. Software companies now realize that software security creates a competitive advantage to market their products (Barnum & Sethi, 2006; Devanbu & Stubblebine, 2000).

Security requirements can no longer be ignored. The increasing number of software security threats combined with general security awareness means that software security is no longer an additional feature, but an expectation. Consider the analogy of bank security. A customer walking into a bank has an assumed expectation of security. They expect security via safes, locks, guards, and identity verification. These basic security devices are easy to understand and can be verbalized regardless of technical expertise. There are likely to be additional security devices in place at a bank, but understanding these devices requires additional technical expertise that the average customer does not possess. While customers do explicitly request all elements of banking security, they express their requirements by choosing the bank with a combined fee and security structure that balances their needs. Consumers of software have similar security appetites. Security may again be expected, but verbalizing specific security requirements may be difficult due to a lack of understanding. It is difficult to elicit security requirements without the aid of those experienced with software security. Justifying additional costs for security, in terms of time or money, can be a difficult sell since they are non-functional requirements.

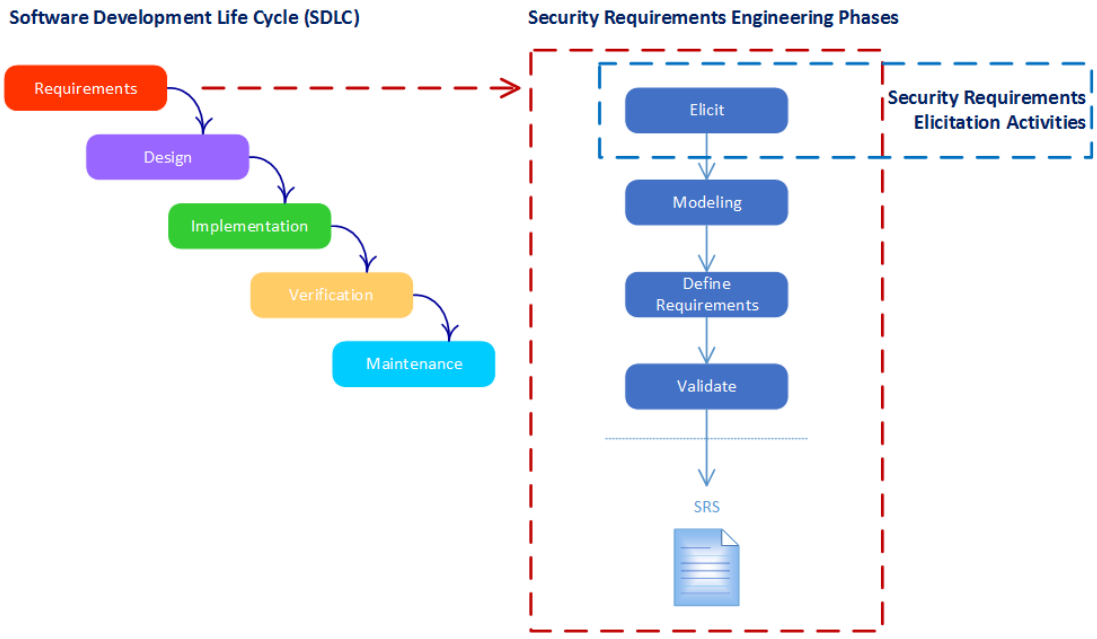

Figure 1.1: Security Requirements Elicitation Approach

We propose a security requirements elicitation approach that is part of the requirements elicitation phase (see Figure 1.1). Preliminary functional requirements artifacts are used as inputs and draft security requirements are output. Although not part of the approach, draft security requirements can be then modeled, defined and validated as part of the final software requirements specification (SRS). The security requirements elicitation approach activities are defined as follows:

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

An overview of the security requirements elicitation approach is shown in Figure 1.2. The tasks comprising our proposed approach are discussed in Chapter 3.

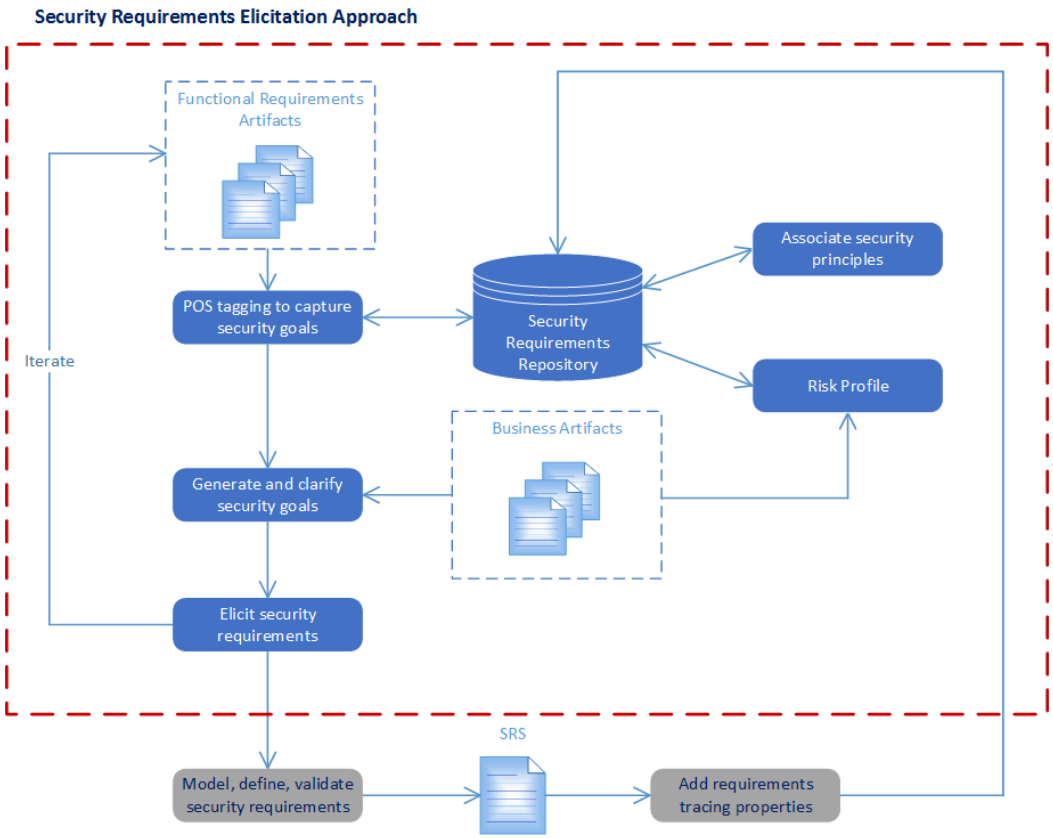

Figure 1.2: Security Requirements Elicitation Approach

1.3 Research Methodology

The security requirements elicitation approach was evaluated and validated using an empirical research methodology. Existing software requirements approaches were studied to understand the current state. Best practices, frameworks, methodologies, models and elicitation techniques were examined to determine applicability to a small, agile organization. Next, a small, agile software development organization

was studied. Observations regarding organizational roles, agile development processes and security requirements development practices were made. These observations combined with the study of software requirements approaches formed the basis for areas to be addressed by the proposed solution. A unique approach using part of speech tagging, analysis tools and a security requirements repository were modeled for the approach. Activities, roles and sample artifacts were developed. Next, the approach was experimentally evaluated with representative organizations. The experimental results were evaluated in order to validate the effectiveness of the approach solution.

1.4 Thesis Organization

This thesis is organized into the following chapters:

- **Chapter 1: Introduction** – The background of the problem, significance, and research methodology for the solution.
- **Chapter 2: Previous Work** – A survey of software security requirements approaches.
- **Chapter 3: Security Requirements Elicitation Approach** – POS tagging, security requirements repository and activities for the security requirements elicitation approach.
- **Chapter 4: Research Results Evaluation and Validation** – The security requirements elicitation approach evaluation and validation.
- **Chapter 5: Conclusion** – Conclusion and recommendations for future research.

2 Survey of Software Security Requirements Approaches

Approaches to software security requirements engineering are evolving to address the lack of integration into existing development processes. General software requirements engineering methods have evolved over time, but little research focuses on the specific aspect of software security requirements. Given that attention to software requirements has been studied and shown to be beneficial to project budget and defect reduction (Mead & Stehney, 2005), additional attention to software security requirements should also prove beneficial.

Security is traditionally classified as a non-functional requirement and many of the approaches for developing security requirements have roots in software quality (Haley, Laney, Moffett, & Nuseibeh, 2008; Mead & Stehney, 2005). Improved quality leads to reduced defects and lower software development costs if conducted early in the development process (Mead & Stehney, 2005). If a software product is not secure, it can be seen as defective. A secure software product may therefore been seen as of higher quality if it has fewer security defects. Quality and security are also similar in that defining each in terms of a software product can be challenging. Security, like quality, is in the eye of the beholder.

Quality and security defects are similar in that a tradeoff analysis may determine acceptable levels (i.e., the level of acceptable quality or security). A key difference between quality and security is that quality defects can be viewed as

unintentional whereas security defects are the result of intentional action or inaction. A business would rarely intentionally produce a product or service that is seen as having a quality defect if they wish to remain in business. On the other hand, security defects may not be addressed during development due to prioritization of requirements that do not leave room for addressing security issues. Other security defects may simply not be known to the stakeholders and are therefore unintentionally omitted. An attacker specifically targets a security defect for some malicious purpose. Software engineers do not intentionally introduce defects into software, yet about half of defects leading to security vulnerabilities found in today's software are actually attributable to flaws in architecture and design (Allen, Barnum, Ellison, McGraw, & Mead, 2008). Security vulnerabilities tend to be the result of the ad-hoc nature of incorporating security into the development process rather than taking a proactive approach from the beginning of the process (Malone & Siraj, 2008).

A lack of education and understanding are contributing factors to the ad-hoc nature of incorporating security requirements into software. Generating software security requirements can be difficult given that it is necessary to have a "black hat" mentality (i.e., thinking like an attacker) when maliciously exploiting a vulnerability. Requirements are derived based on what a software product should do, not what it should not do. In order to derive good security requirements, software engineers must be educated and experienced in all aspects of software security. However, in practice, this may not necessarily be the case since education in security is often lacking

(Barnum & Sethi, 2006; Viega, 2005) and a "security by obscurity" (Barnum & Sethi, 2006; Mercuri & Neumann, 2003) mentality persists among software engineers. Hiding or attempting to obscure software flaws in the hope that attackers will not find them is a poor approach to security. Educating software engineers to think like an attacker will improve the proactive integration of security requirements into the software development lifecycle.

Abuse cases, misuse cases, attack trees and security patterns are approaches designed to develop "black hat" thinking among software engineers and stakeholders. Developing misuse and abuse cases is an example of an approach that aids both software engineers and users to see beyond expected use in order to develop security requirements. The case approach encourages user input and aids all stakeholders with visualizing different scenarios. Attack trees and security patterns are different approaches to visualize scenarios but are presented in a different manner. All of these approaches encourage user input and interaction with software engineers which is critical when eliciting and developing software requirements. User security awareness should also improve when these approaches are implemented. Users expect security, but have difficulty defining security and should not be the sole source of developing security requirements. Broadly, users may define security requirements as meeting state and federal regulations or upholding company policies without a clear idea of what the regulations or policies entail. Therefore, the "black hat" approaches that software engineers use can aid users in defining security requirements.

Another tactic that may be followed is to test security into software products (McGraw, 2005). Although not an ad-hoc approach, testing security into software is certainly not a proactive approach either. Traditionally, testing in software development life cycles occurs near the end of the project. If security flaws are uncovered during testing, significant rework may be required, especially if testing is in the later stages of development. Even agile organizations which integrate testing throughout the development lifecycle will experience rework if security requirements are discovered during testing rather than being specified at the start. In fact, security cannot conclusively be proven through testing regardless of the development process (McGraw, 2005). For example, how is it possible to conclusively test against the unauthorized disclosure of information? If security cannot be "tested in", security requirements must be developed along with functional requirements in order to meet stakeholder goals.

To understand the current nature of software security requirements approaches, best practices, enumerations, frameworks, methodologies, elicitation techniques and models were studied. Best practices and enumerations range from very broad activities to focusing on specific solutions. Frameworks and methodologies expand the view and begin addressing the security requirements process as a whole. Elicitation techniques focus on defining or drawing out key elements under consideration. Models define relationships between elements in a structured manner. Understanding the current state of security requirements approaches can be useful in determining barriers and drawbacks. Based on this

survey of approaches, the following sections will outline the evolution of current security requirements approaches. Pros and cons to current approaches will be addressed in order to determine the key elements required to develop a new security requirements elicitation approach.

2.1 Best Practices and Enumerations

Software security engineering has evolved from best practices such as enforcing coding standards to mitigating risk at the organizational level (Giorgini, Massacci, Mylopoulos, & Zannone, 2005). Best practices are often associated with the design and implementation phases rather than during the requirements specification phase (Falcarin & Morisio, 2004; McGraw, 2008). Best practices should enhance requirements elicitation and analysis during requirements specification instead of focusing purely on design and development phases. SSDL Touchpoints and OWASP cheat sheets are best practices approaches that were examined to determine relevancy to the requirements specification and analysis phases of development.

2.1.1 SSDL TouchPoints

Secure Software Development Lifecycle (SSDL) Touchpoints are part of the Software Security Framework (McGraw, Migues, & West, 2012). SSDL Touchpoints consist of architectural analysis, code review and security testing practices which should be included in any software security framework. Architectural analysis occurs early in the development lifecycle but is performed after requirements have been specified. Code review and security testing occur even later in the lifecycle. Each of these practices focuses on later stages of development rather

than on earlier requirements elicitation and development phases. Abuse cases and attack patterns are recommended practices to be used during the requirements phase, but specific details are not given as part of the framework (McGraw, 2005). Touchpoints provide an overview of practices that should be followed but do not define specific tasks or processes for accomplishing these practices. Therefore, SSDL Touchpoints broadly address all areas of development rather than specifically focusing on requirements elicitation.

2.1.2 OWASP Cheat Sheets and Enterprise Security API

"Cheat sheets"[1], such as those available at The Open Web Application Security Project (OWASP), are intended to aid software engineers with solutions to specific security problems and as overall guidance for application security. Compiled by an open source community of security experts, OWASP cheat sheets tend to target specific development activities and provide tips in the form of "what-to-do" and "what-not-to-do". The majority of the cheat sheets give guidance on development specific topics rather than requirements development. For example, the authentication cheat sheet provides general guidelines related to passwords including length, complexity, secure recovery mechanisms and authentication error messages. General examples of security vulnerabilities related to authentication are given, but attacker scenarios, such as abuse cases, are not covered in detail. Many of the cheat sheets target later stages of software development rather than the requirements elicitation phase.

[1] `https://www.owasp.org/index.php/Cheat_Sheets`

Cheat sheets are dynamic documents and can be categorized as either established or draft versions. Established cheats sheets primarily address code development activities, but draft cheat sheets are being developed that take a broader approach. Secure SDLC and threat modeling are topics under development that may prove more useful for requirements development activities rather than later stages such as design and testing. As a requirements elicitation tool, cheat sheets provide an opportunity to open the discussion with stakeholders on application security related topics.

The Enterprise Security API (ESAPI)[2] is a security control library for web application development. The ESAPI is downloadable for several development languages, but the extent of each library varies significantly. As a supporting development tool, the ESAPI is best used to implement security requirements, but not as a requirements elicitation or development tool.

There are disadvantages to using OWASP "cheat sheets" as a security requirements approach. As the name implies, the OWASP community focuses on web applications. Security principles that apply to web applications are transferrable to other types of applications, but as a general approach to developing software, OWASP may be limited in scope for some software development projects. OWASP is also developed and maintained by an open source community with loose affiliation to software development organizations. Ongoing support and resources are limited to the enthusiasm of the community and project priorities vary. Finally, OWASP tends

[2] https://www.owasp.org/index.php/ESAPI

to focus on later stages of the SDLC rather than the early requirements phase. These factors may limit the OWASP resources as a viable security requirements approach beyond increasing security awareness during requirements elicitation activities.

2.1.3 Enumerations and Classifications

The U.S. Department of Homeland Security office of Cybersecurity and Communications co-sponsors enumeration and classifications sites for cybersecurity related topics. Sites are publically available and are sponsored by the MITRE Corporation. A community of individuals and organizations maintains and develops each site based on their respective interest in the site. Three sites are commonly cited in the field of software security:

- Common Weakness Enumeration (CWE)[3]
- Common Vulnerabilities and Exposures (CVE)[4]
- Common Attack Pattern Enumeration and Classification (CAPEC)[5]

The CWE provides a description of more than 700 software weaknesses each with applicable platform, common consequences and examples. The CVE site provides a database of vulnerabilities and exposures identified by participating organizations. CVE identifiers are numbered and include brief information about the vulnerability and exposure. Vulnerability scanners and reporting tools can use the CVE identifier to provide feedback to the developer for further analysis. Both the CWE and CVE provide technical information that requires knowledge of software systems and is unlikely to be understood by the average user. Specific topics can be searched in

[3] http://cwe.mitre.org/
[4] http://cve.mitre.org/
[5] http://capec.mitre.org/

either database, but a basic level of security knowledge is required as a starting point. For example, a specific weakness or vulnerability would have to be known first in order to search either database. The amount of detail provided varies and specific information that could be used for risk analysis is not included. Details provided are also generally focused on later stages of development, such as design, testing and maintenance.

The CAPEC provides attack pattern information in a format similar to CVE and CWE and lists over 400 attack patterns. CAPEC identifiers include a general description, attack prerequisites, likelihood of exploit, methods of attack and examples. A scope of attack motivation and consequences (i.e., loss of confidentiality, lack of authorization) is given which could be used to understand general security principles, but this small detail is lost with respect to the other detail presented for each attack pattern. Like the CVE and CWE, the information is presented in a very technical nature and does not include easily interpreted diagrams. Therefore, enumerations and classifications are of limited use as tools for requirements elicitation. Additional discussion of the use of attack patterns beyond the CAPEC classification is included in subsequent sections.

2.2 **Frameworks**

Frameworks are a general structure in which the user can choose to define the actual approaches used to solve a problem. Secure TROPOS, the Software Security Framework, the Building Security In Maturity Model, and Integrating Requirements and Information Security were surveyed.

2.2.1 Secure TROPOS

Secure Tropos extends the Tropos methodology to develop a "formal framework for modeling and analyzing security and trust requirements" (Giorgini, Massacci, Mylopoulos, & Zannone, 2004). Tropos takes into account not only computer systems, but the organizational environment in which the system interacts. This includes the perspective of the users and stakeholders (actors) as well as their interactions, goals, shared resources and dependencies. Secure Tropos incorporates security requirements engineering by modeling and analyzing trust and delegation relationships among agents or services. The trust model has similarities with abuser stories which are common in agile development.

A drawback to using Secure Tropos is the assumption that requirements been elicited and identified in order to be modeled and analyzed. Dependency and trust models need to be created. If these models are complex, the requirements engineer would need to spend significant time and resources developing the model along with taking time to explain and clarify the model to business stakeholders. The model also does not include risk information that could be used for prioritization of goals. Other tools, such as abuser stories, may be easier to use and implement during the requirements elicitation phase for a project with limited resources.

2.2.2 The Software Security Framework

The Software Security Framework (SSF) addresses overall security, not just the development of software security requirements (McGraw, Migues, & West, 2012). SSF is organized into four domains: Governance, Intelligence, SSDL Touchpoints, and Deployment. Each domain has three practices with individual activities (111

total activities for all domains). The domains cover overall organizational security activities, but the intelligence domain addresses software security activities. Intelligence domain practices include attack models, security features and design, and standards and requirements. Key elements from the intelligence domain can be useful when eliciting security requirements. In particular, attack models can aid software engineers in eliciting security requirements by encouraging them to think like an attacker. Although SSF defines specific practices to address security requirements engineering, the large number of activities and abstract nature of the framework do not make SSF suitable as a requirements elicitation solution

2.2.3 Building Security In Maturity Model

The Building Security In Maturity Model (BSIMM)[6] was developed from security initiative data gathered from fifty-one organizations (McGraw, Migues, & West, 2012). The McGraw study began in 2008 and has evolved over the years to the current fourth iteration. McGraw maintains that developing only security processes is insufficient and a broader security initiative is required. The current iteration, BSIMM4, incorporates four domains with 12 practices for a total of 111 activities. An organization can assess their efforts for these activities to create an overall security score. Scoring allows an organization to compare internal efforts with peer organizations.

Although the name implies that BSIMM is a specific model that can be followed, it is actually part of the SSF. This allows organizations to choose specific

[6] http://www.bsimm.com

models or activities they wish to follow when undertaking a security initiative. BSIMM is intended to be used by organizations to benchmark current security activities against organizations participating in the yearly study. Based on assessment from the domains, an organization can choose any model they deem appropriate to address deficiencies. McGraw describes BSIMM as a "descriptive model" rather than providing "prescriptive guidance" (McGraw, Migues, & West, 2012).

An advantage of BSIMM is that it provides an organization with broad security perspectives to build an initiative. Deficiencies in any practice area or domain can be prioritized to improve the security maturity level for the organization. The disadvantage is that the organization must still choose an approach to addressing deficiencies. BSIMM gives overall guidance in improving security initiatives for an organization of which software development activities are included in the intelligence domain. Attack models are the key activity in the intelligence domain and can be used to supplement requirements elicitation activities. Other activities include standards and requirements, but general guidance is given without a specific framework to follow to complete these activities. Therefore, BSIMM does not provide enough detail to be used primarily for security requirements elicitation and development.

2.3 **Methodologies**

2.3.1 SQUARE

The SQUARE model was initially developed as the System Quality Requirements Engineering model by researchers at Carnegie Mellon and the Software Engineering

Institute (Mead & Stehney, 2005). The elicitation and prioritization phases of software development are the focus of the methodology. Nine steps are defined (see Table 2.1) which produce an output based on recommended input information. Each step has defined example techniques to accomplish each step as well as likely stakeholder participants. A CASE tool has been developed by Carnegie Mellon researchers to implement the methodology (CERT-SEI, 2010).

A possible drawback to the SQUARE model is in step three (develop artifacts). Researchers suggest that these artifacts may be related to the design phase rather than the requirements phase (Tondel, Jaatun, & Meland, 2008). The methodology is quite complex and lengthy. Requirements specification may take months and requires considerable resources to use the methodology. For small software development projects, it is likely that entire projects would be completed during this time frame. Therefore, a more flexible, efficient approach requiring fewer resources is desired.

Table 2.1: SQUARE Methodology Steps (Mead et al., 2005)

Step	Description
1	Agree on definitions
2	Identify security goals
3	Develop artifacts to support security requirements definition
4	Perform risk assessment
5	Select elicitation techniques
6	Elicit security requirements
7	Categorize requirements as to level (system, software, etc.) and whether they are requirements or other kinds of constraints
8	Prioritize requirements
9	Requirements inspection

2.3.2 CLASP

Secure Software's CLASP (Comprehensive, Lightweight Application Security Process) was developed in 2005 to address all processes for software security development from requirements through testing and deployment (Viega, 2005). The Open Web Application Security Project (OWASP) website[7] states that CLASP "contains formalized best practices" related to all aspects of software development. CLASP is intended to be applicable to existing software or new development projects using high-level perspectives or views. CLASP views include concepts, roles,

[7] CLASP downloads available at:
http://www.owasp.org/index.php/OWASP_CLASP_Project#Downloads

activity assessment, activity implementation and vulnerability. Views tend to cascade down starting with the concepts view, but views that may be revisited and cycled back through in a continuous improvement manner. The iterative nature CLASP departs from traditional development and favors agile development.

CLASP is not a one-size fits all solution for improving application security. The 24 CLASP activities are not mandatory, but can be addressed at the discretion of the implementing organization. Metrics for choosing and prioritizing activities are not specified, this is left up to the organization to choose. In addition, many of the activities are recommended to be carried out with the use automated tools, but specific tools are not defined. Details of the problem types for the vulnerability view are provided in a static checklist which can aid in the implementation of CLASP, but is not in the form of an easy to use tool. CLASP is therefore a generic roadmap which can be used for all software development activities, not a step-by-step checklist.

2.3.3 OCTAVE

Operationally Critical Threat, Asset and Vulnerability Evaluation (OCTAVE)[8] is a risk assessment methodology developed by the Software Engineering Institute at Carnegie Mellon. Several tools are available for implementing OCTAVE and three OCTAVE methods are available. Smaller organizations can use the OCTAVE-S method for a less intensive approach requiring fewer resources. OCTAVE is not a security requirements development approach. It is part of a larger initiative that an

[8] http://www.cert.org/octave/octaves.html

organization can undertake to assess organizational risk. The results of the risk assessment can be used by an organization to understand broader security initiatives or to improve security awareness. If an organization has undertaken risk assessment, the results may be useful in directing the development of security requirements. OCTAVE is not a lightweight approach that would be useful to undertake during requirements development for small, agile organizations.

2.3.4 USeR Method

Usage-centric Security Requirements engineering (USeR) method integrates quality tools into requirements engineering to extract security requirements from software requirements (Hallberg & Hallberg, 2006). Voice of the customer (VoC) is a quality term often associated with quality function deployment and Six Sigma that helps to define the quality viewpoint of the customer (Gitlow & Levine, 2005). USeR implements a voice of the customer table (VCT) by selecting requirements that appear related to security and generating security statements. Each statement is analyzed by asking who, what, when, where, why and how to further understand each security statement. Security needs and the resulting security requirements are further processed using affinity and hierarchy diagrams. Two additional processes are performed to analyze security techniques and design implications.

The USeR method is an approach to extract security requirements when users have a hard time explicitly defining security. Stakeholders take an active role in the VCT analysis but must be guided by a skilled facilitator to fully extract security needs. Details of techniques to construct affinity and hierarchy diagrams are lacking

and security expert knowledge is assumed to prioritize the diagrams. The researchers note that tools such as misuse cases may be needed to improve requirements visualization. However, the concept of extracting security requirements from general requirements and using quality techniques intriguing and aligns with the goals of this thesis. Different techniques may be implemented to improve the USeR method as an improved security requirements approach.

2.3.5 SURE/ASSURE

Secure and Usable Requirements Engineering (SURE) and Automated Support for Secure and Usable Requirements Engineering (ASSURE) propose to provide support throughout out all stages of the software development lifecycle (Romero-Mariona, 2009). SURE builds on previous approaches to improve the development of security requirements and increase the usability in subsequent development activities. There are two main steps for the SURE process: security requirements and security testing: The security requirements process combines existing approaches by implementing CLASP activities and the USeR method. Security statements evolve to security needs and then finally, security requirements are created (Hallberg & Hallberg, 2006). Security testing focuses on later stages of software development by deriving three sets of test cases. Misuse cases and threat consequences are modeled as inputs for the test cases.

The process of developing security requirements is an extension of the USeR method that also specifies misuse cases (see section 2.4.4 for further discussion of misuse cases). Furthermore, the primary focus of SURE is to support all stages of

software development rather than just focusing on requirements specification. As discussed in previous sections, the USeR method is an intriguing approach to extracting security requirements. The proposed requirements elicitation approach expands on the general concepts of the USeR and SURE methodologies to improve security requirements elicitation in an iterative nature.

2.4 Elicitation Techniques and Models

Elicitation techniques and models are often used in conjunction with each other during the development process. Commonly, elicitation techniques are activities that assist stakeholders with defining security requirements. The artifacts developed from these techniques are then incorporated into the model for further analysis. Models are used for different purposes during software development. Some models use software requirements as input in order to design the system architecture. Other models are used to during elicitation activities to develop requirements. Elicitation techniques and models to address security in software projects come in varying forms. Some are geared toward security experts; others do not assume expert knowledge. The following sections discuss common elicitation techniques and models that are geared towards software security.

2.4.1 UMLsec and SecureUML

UMLsec extends the Unified Modeling Language (UML) to specifically model security features (Jürgens, 2002). Security profiles are generated consisting of a concept called stereotypes that include tagged values and constraints. A goal of UMLsec is to aid software engineers who do not have strong security backgrounds to

use UMLsec to model security requirements (Jürgens, 2001). Automated tools can be used to implement security checking based on UMLsec (Falcarin & Morisio, 2004). Like UMLsec, SecureUML is a security modeling language which also extends UML to include specific security constraints related to access controls and is based on role-based access control (RBAC) security model (Lodderstedt, Basin, & Doser, 2002). The main drawback to SecureUML is that the primary focus is to aid in design activities rather than requirements specifications.

A drawback to both UMLsec and SecureUML is the assumption that software engineers have a background with UML and will be able to quickly incorporate security modeling into UML diagrams. The formal nature of UML diagramming works best in traditional development but could be a drawback for agile development teams. To address audiences beyond UML users, CARiSMA[9] is a newer security modeling tool that is integrated into the popular Eclipse IDE. CARiSMA was built to succeed UMLsec, but it appears that this takes the emphasis even further from requirements specification and deeper into the design phase. In addition, UMLsec has a strong focus on critical systems development which may limit usefulness as a general security modeling tool.

2.4.2 SDL and STRIDE

The Microsoft Security Development Lifecycle (SDL)[10] is a group of security practices that can be integrated into the software development lifecycle. Practices are grouped into training, requirements, design, implementation, verification, release and

[9] http://vm4a003.itmc.tu-dortmund.de/carisma/web/doku.php
[10] http://www.microsoft.com/security/sdl/default.aspx

response phases. Tools are available for most phases, but are very broadly defined. For example, there are three practices for the requirements phase. The security requirements practice generally states that security and privacy requirements should be defined early. However, no practical mention is made of how to go about defining security requirements. The SDL provides useful guidance for an overall security development initiative, but is lacking in detail. Downloadable tools, where available, also make the assumption that development takes place using Microsoft Visual Studio.

Threat modeling is treated as a design phase practice. STRIDE is a threat modeling approach developed by Microsoft to be incorporated during design. STRIDE stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege. Each of these threats is tied to a specific security property (i.e., confidentiality, integrity, availability, etc.). Unlike other models which focus on assets or attackers, SDL focuses on overall software development using a tool based approach designed for ease of use by software engineers. The SDL threat modeling process is illustrated as a cyclical process with activities of diagram, identify threats, mitigate, and validate making it similar to other continuous improvement processes. Compliance with Microsoft SDL process includes but is not limited to 16 mandatory security activities. A drawback of STRIDE is that language and concepts covered tend to target software engineers rather than a larger group of stakeholders that includes non-technical users. In fact,

the STRIDE overview emphasizes use during the design phase with software engineers and architects as the primary audience.

2.4.3 Extended Activity-Based Quality Model

The concept of security is similar to quality in that while difficult to define, once agreed upon, security concepts can be reused. Subsequent reuse of requirements will lead to a reduction of project cost. This concept is the emphasis behind the Extended Activity-Based Quality Model (eABQM) which is implements reuse of security requirements (Luckey, Baumann, Méndez, & Wagner, 2010). Security requirements are modeled as facts and activities both of which are dependent on impact. The researchers theorize that incorporating project goals, parameters and relevance factors into the model will support requirements reusability. The ability to model security requirements and the reusability of results were the primary goals of implementing eABQM.

This approach does not support the initial generation of security requirements, but does show promise that the development of a security requirements repository can reduce project cost. While not discussed, a requirements repository could aid software engineers in improving security awareness and training leading to overall improvements in developing secure software. The concept of project parameters and categories could be modified to be security terminology and categories for an improved approach.

2.4.4 Misuse Cases, Security Use Cases and Abuse Cases

Misuse cases are a negative form of use cases used to elicit non-functional security requirements and analyze security threats (Alexander, 2003; Firesmith, 2003).

Misuse cases are generated to clearly highlight how a misuser can violate application security (Firesmith, 2003). In this context, a misuser is generally defined as an insider or an outsider with malicious intent. Firesmith refines the use case concept to analyze and specify security requirements through the development of security use cases. Software engineers create misuse cases which drive the development of security use cases to be used to develop security requirements. Reusability of security use cases is more feasible if they are as generically defined as possible with details abstracted out (Firesmith, 2003). Architectural and design decisions should not be made when developing security use cases. Security requirements should also avoid specifying security mechanisms. Abuse cases (also referred to as threat scenarios) should be written in the stakeholder's language to ensure understanding (Boström, Wäyrynen, Bodén, Beznosov, & Kruchten, 2006). Abuse cases are created in a form in which the interactions of actors results in a security violation. In this context, abuse cases differ from misuse cases in that interactions that should not be allowed are modeled in abuse cases (Giorgini et al., 2004).

2.4.5 Abuser Stories

User stories are commonly used in agile processes to capture the user's requirements for the product under development and are preferably written by users. This presents a quandary when developing security requirements. The agile development team most likely does not have an abuser (i.e., hacker, attacker) on the team nor are they considering what the system should not do (attacks). Abuser stories are intended to take into account the attacker perspective in order to develop security requirements

for the proposed system (Peeters, 2005). User experiences can be useful in determining past security mishaps that may not be readily apparent to the development team. Because user input is critical to creating user stories, this reinforces the need for ongoing security awareness and training for all stakeholders in order to anticipate security concerns and develop security requirements.

Similar to user stories, abuser stories are ranked and scored by business value but also include perceived threats posed to assets (Peeters, 2005). Prioritization of security goals plays a key role in developing security requirements for the approach proposed in this thesis. Data used for ranking abuser stories can be used as input for the prioritization and validation activities. As an agile elicitation technique, user stories are easy to implement and do not require significant training. The concept of generating a prioritized set of security goals is an intriguing concept that will be useful for the proposed approach. Providing a method to prioritize security goals and iteratively creating security requirements could be an improvement to the concept of abuser stories.

2.4.6 Attack Trees

Decision trees are commonly used to graphically demonstrate the route and processes required to reach a decision. Modeling the decisions that attackers make on a system has evolved into the concept of attack trees. These trees are used to graphically analyze attack scenarios and incorporate cost or probability statistics so that the threats can be prioritized. The root of the tree represents the attacker's goal and the

leaves represent possible avenues to achieve the goal. Highly detailed trees can be created, but the detail may be limited to the existing knowledge of the tree developer.

Attack trees can go beyond relying on developer intuition about attack vectors to formalize risk analysis from the viewpoint of the attacker. (Ingoldsby, 2009) Attack scenarios are created and resources needed (e.g., attacker's cost and ability) are used to establish scenario costs. The difficulty that software engineers may have in determining cost is that they must be experts in the area to determine resource requirements and they must put on their "black hats". When creating initial attack trees, some attack scenarios may have an unlikely probability of being carried out and are "pruned" or removed from the tree. Attackers make cost-benefit decisions just as software engineers make the same decisions when prioritizing security requirements. SecurITree[11] is a commercially available attack tree tool that can be used to facilitate attack tree modeling.

Attack trees are another model that can be used for risk and cost-benefit analysis, but software engineers still need to have a considerable arsenal of information available to begin constructing attack trees. The model provides a more structured approach than using best practices and checklists, but still requires education as well as the ability to apply statistical information. Small, agile organizations simply may not have the resources to implement attack tree analysis as it requires security experts for statistical and cost analysis.

[11] http://www.amenaza.com/

2.4.7 Attack Patterns and Security Patterns

Patterns are developed through the application of knowledge or experience in order to be reused over and over again. Over time, adjustments are made to an existing pattern to improve the pattern or to develop a new pattern. "Attack patterns describe the techniques that attackers may use to break software" (Barnum & Sethi, 2006). The concept of patterns for software development originated with design patterns for reusability. Attack and security patterns expand on this concept by attempting to build catalogs of patterns to close the gap between attackers and software engineers. Cataloged information includes pattern name/classification, attack prerequisites, related vulnerabilities/weaknesses, attack methods (vectors), knowledge required, and recommended solutions. Security patterns were proposed to "bridge the gap" between security professionals and systems developers. The Common Attack Pattern Enumeration and Classification[12] (CAPEC) was developed by and are supported for the larger software development community to aid in secure software development.

2.5 **Comparison of Approaches**

Integrating security requirements into the software development process can be difficult. Development teams who have not previously considered security requirements will not only need to integrate new processes into existing development but also to understand an entirely new set of problems. Understanding security terminology, existing and emerging vulnerabilities, analyzing security risk and prioritizing security requirements into ongoing processes may be difficult. Software

[12] `http://capec.mitre.org`

engineers will seek manageable, efficient approaches to ease this transition. This survey highlights some of the proposed approaches to address specific issues for security requirements development.

An organization undertaking a security initiative may be overwhelmed when trying to determine where to start. The approach taken may depend on many factors including the size of the organization, security awareness, security culture, types of products developed, education levels of software engineers and software development processes followed. A security event may have thrust the initiative into high-gear. Perhaps there was a general security breach, a pending proposal for a new product requiring security features or pending legislation. Regardless, the organization determines that security requirements are to receive attention. The prioritization of integration may require quick action or as part of an ongoing effort. All of these factors should be considered as improvements to security requirements integration approaches.

None of the existing approaches focuses on the financial or risk aspects based on new legislation. It is difficult to keep up with pending and new legislation as well as to understand due diligence required to meet regulations. The impact of regulation should be part of the early requirements elicitation process. This will help users and software engineers understand general security vulnerabilities that may not have been given prior consideration as well as to place financial impact on determining which requirements to include. It is not feasible to include all features that a user desires whether it is a general feature or security specific feature. Including risk cost analysis

if a security requirement is not implemented will clarify security issues for all stakeholders.

Software engineers are likely to have requirements documents processes in place. The prototype approach should include an automated scanning tool, much like automated code review tools, to identify areas to include security requirements. For example, requirements may already be included that are not specifically identified as security requirements, but that include security components. Scanning for security related terminology and identifying these requirements could jump start security integration process. Stakeholders will have a starting point for beginning to increase security awareness rather than facing a multi-step approach that may not yield results as quickly as desired.

Finally, the surveyed approaches are either broad based or focus on specific phases of the SDLC (see Table 2.2). Focusing on only one aspect of development process, especially later stages, can lead to omitting important elements. Security requirements could then become less important if they are seen as part of the process that someone else will take care of during later phases of development. The same principle occurs with quality. If quality is seen as something to be checked at the end of the process, the natural instinct is to pass the problem down the line. However, when quality becomes the concern of the entire process, then quality is prioritized and is built into the entire process. The same concept should be applied to the integration of security requirements into the entire process.

Table 2.2: Comparison of Security Requirements Approaches

Approach	R	D	I	V	M
Best Practices and Enumerations					
SSDL Touchpoints					
OWASP Cheat Sheets					
CWE, CVE, CAPEC					
Frameworks					
Secure Tropos					
SSF					
BSIMM					
IRIS					
Methodologies					
SQUARE					
CLASP					
OCTAVE					
USeR					
SURE/ASSURE					
Elicitation Techniques and Models					
UMLsec and SecureUML					
SDL					
STRIDE					
eABQM					
Misuse, security use. abuse cases					
Abuse stories					
Attack trees					
Attack patterns					
Security patterns					

Software Development Design Phases

R = Requirements

D = Design

I = Implementation

V = Verification

M = Maintenance

Strong association

Weak association

3 Security Requirements Elicitation

Research has shown that integrating security requirements into the early phases of the software development life cycle has significant benefits (Mead et al., 2005; Moffett, Haley, & Nuseibeh, 2004). 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, frameworks, methodologies, elicitation techniques and models have been proposed that are intended to improve the integration of security requirements into early phases of development. SSDL Touchpoints, SSF, BSIMM, and OWASP take a very broad view emphasizing building security initiatives at all stages of software development. SQUARE, CLASP and Secure Tropos address integration of security requirements but are geared towards long development lifecycles and could be cumbersome for agile organizations. IRIS and SURE/ASSURE seek to improve usability of security requirements rather than eliciting security requirements. OCTAVE and STRIDE are used for threat modeling. CWE, CVE, CAPEC aid software engineers during design and coding phases to implement requirements rather than eliciting requirements. UMLsec, SecureUML and eABQM model security features and support reuse of requirements. Finally, misuse cases, abuse stories, attack trees and other approaches are elicitation activities that can be undertaken as part of a larger security requirements elicitation initiative.

Each of these approaches may be useful when developing security requirements, but are too broad, too specific, lengthy, or require expert knowledge to be used with agile software development. Therefore, integrating security requirements into existing software development lifecycles in a manner that can be implemented by small, agile organizations is proposed. The proposed elicitation approach analyzes, prioritizes and develops preliminary security requirements from general software requirements artifacts using POS tagging. USeR and SURE/ASSURE approaches both cite the difficulty with extracting security requirements from users due to a general lack of security knowledge (Hallberg & Hallberg, 2006; Romero-Mariona, 2009). Integrating POS tagging to capture security requirements implied by business stakeholders but not specifically stated is expected to improve security requirements elicitation. The output preliminary security requirements captured from POS tagging can then be modeled, defined, and validated (not covered by this thesis) to generate final security requirements in later stages of the software development cycle. Figure 1.2 gives a broad overview of how the proposed approach integrates into the software development lifecycle.

The proposed requirements elicitation approach will be iterative which will distinguish it from other approaches such as SQUARE (Mead & Stehney, 2005). SQUARE also assumes that a team is designated for the specific purpose of eliciting security requirements. Small organizations simply may not have the personnel resources to allocate an entire team to this task and will need to incorporate these activities into the regular requirements elicitation activities. POS tagging activities

and the implementation of a security requirements repository are also innovative in that they are not currently implemented by the surveyed approaches. The following sections describe the POS tagging approach, security requirements repository, and the identify, categorize, understand and prioritize activities for the security requirements elicitation approach.

3.1 Security Requirements Repository Design

The activities in the security requirements elicitation approach rely on the development of a security requirements repository. A prototype of the security requirements repository is shown in Figure 3.1. The subsequent sections detail the entities and attributes for the repository. (Primary keys are denoted as PK.)

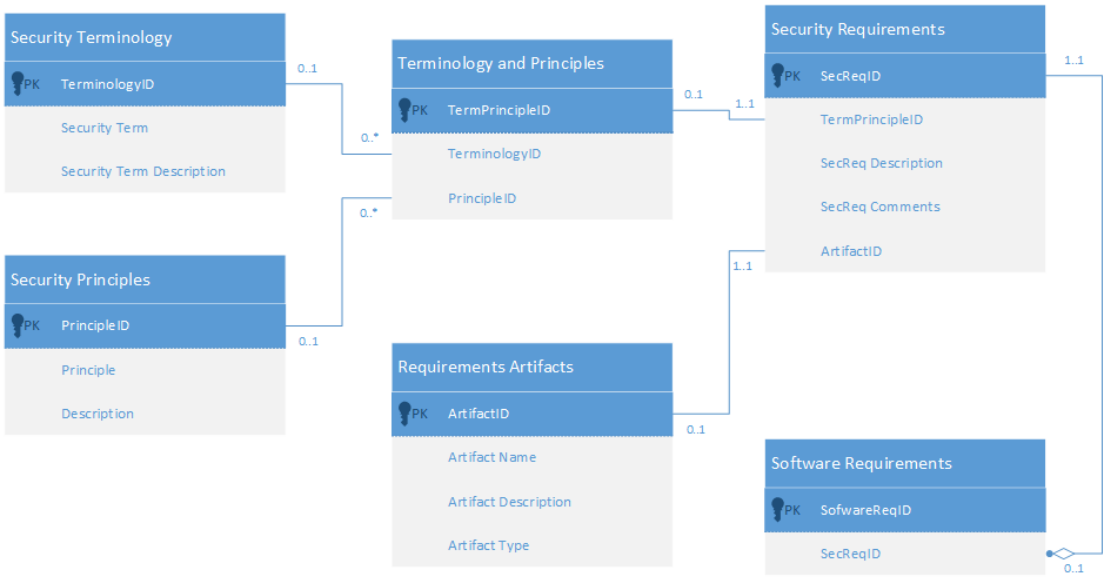

Figure 3.1: Security Requirements Repository Model

3.1.1 Security Terminology Entity

Attributes for the security terminology entity are `TerminologyID` (PK), `Security Term`, and `Security Term Description`. `Security Term` attributes are single terms (unigrams) that will be used during POS tagging. The repository will be populated with terms identified by the requirements engineer based on experience or by using a dictionary of security terms. Over time, security terms can be refined to improve the effectiveness of POS tagging. Each security term has additional details, such as definitions or phrases, which enhance the understanding of each security term. Table 3.1 is a set of security terms that are used to populate the repository prior to POS tagging. These terms were chosen after a broad scan of sample software requirements specification documents for security related terminology.

Table 3.1: Security Terms

Security Terms		
access	certificates	malicious
audit	deny	password
authenticate	encrypt	permission
authentication	encryption	privileges
authorize	https	risk
authorized	logon	security
certificate		

3.1.2 Security Principles Entity

Attributes for the security principles entity are `PrincipleID` (PK), `Principle`, and `Description`. As a minimum, security principles are confidentiality, integrity, and availability (CIA), but additional security principles can be defined as

well. Description attributes are definitions or details to provide a common basis of understanding among stakeholders. Security principles and description for the repository are shown in Table 3.2.

Table 3.2: Security Principles and Description

Principle	Description
Confidentiality	unauthorized disclosure of information
Integrity	unauthorized modification or destruction of information
Availability	disruption of access to or use of information of an information system

3.1.3 Terminology and Principles Entity

Attributes for the terminology and principles entity are `TermPrincipleID` (PK) and secondary keys, `TerminologyID`, and `PrincipleID`.

3.1.4 Requirements Artifacts Entity

Attributes for the requirements artifacts entity are `ArtifactID` (PK), `Artifact Name`, `Artifact Description`, `Artifact Type`.

3.1.5 Security Requirements Entity

Attributes for the security requirements entity are `SecReqID` (PK), `TermPrincipleID`, `SecReq Description`, `SecReq Comments`, and `ArtifactID`. Secondary keys are `TermPrincipleID` and `ArtifactID`. `SecReq Description` is the security requirement that is generated during the elicitation activity. `SecReq Comments` are general comments regarding the security requirement.

3.1.6 Software Requirements Entity

Attributes for the software requirements entity are `SoftwareReqID` (PK) and secondary key, `SecReqID`. The entity relates the newly generated security requirements to the software requirements specification artifact.

3.2 Security Requirements Elicitation Activities

The activities in the security requirements elicitation approach are:

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

Each activity defines inputs, roles, techniques and output. Inputs are requirements related artifacts. Roles are the development team and business stakeholders responsible for the activity. Techniques are applied to accomplish each activity and a security requirements artifact is output. The output for the approach is a prioritized security requirements artifact. Figure 3.2 represents the input, roles, techniques and output for the approach activities.

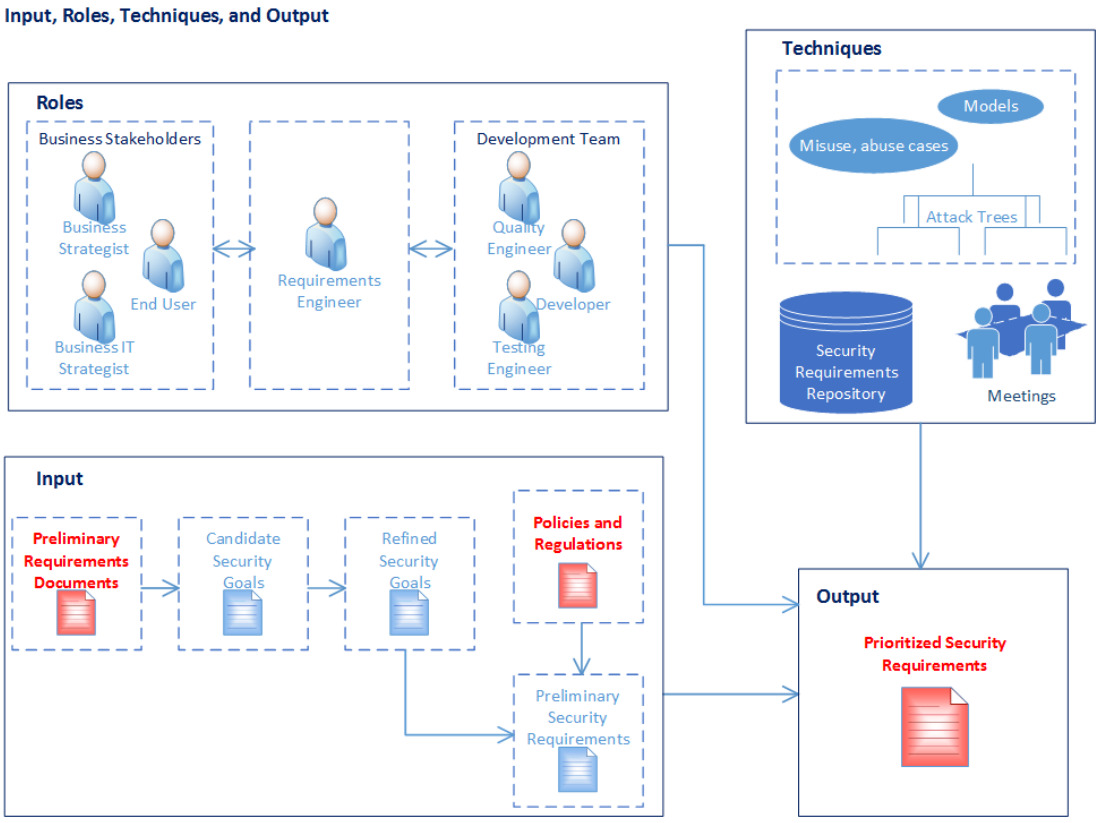

Figure 3.2: Security Requirements Approach Components

3.2.1 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 knowledge of vulnerabilities and threats. However, business stakeholder knowledge about vulnerabilities and threats may not be extensive which leads to ambiguity expressing security needs. The result may be functional requirements written with security terminology that implies security requirements but that are not explicitly defined. If security terminology can be discovered, candidate security goals can be identified that with further analysis could

be used to develop security requirements. Figure 3.3 shows the detail for the identify security goals activity.

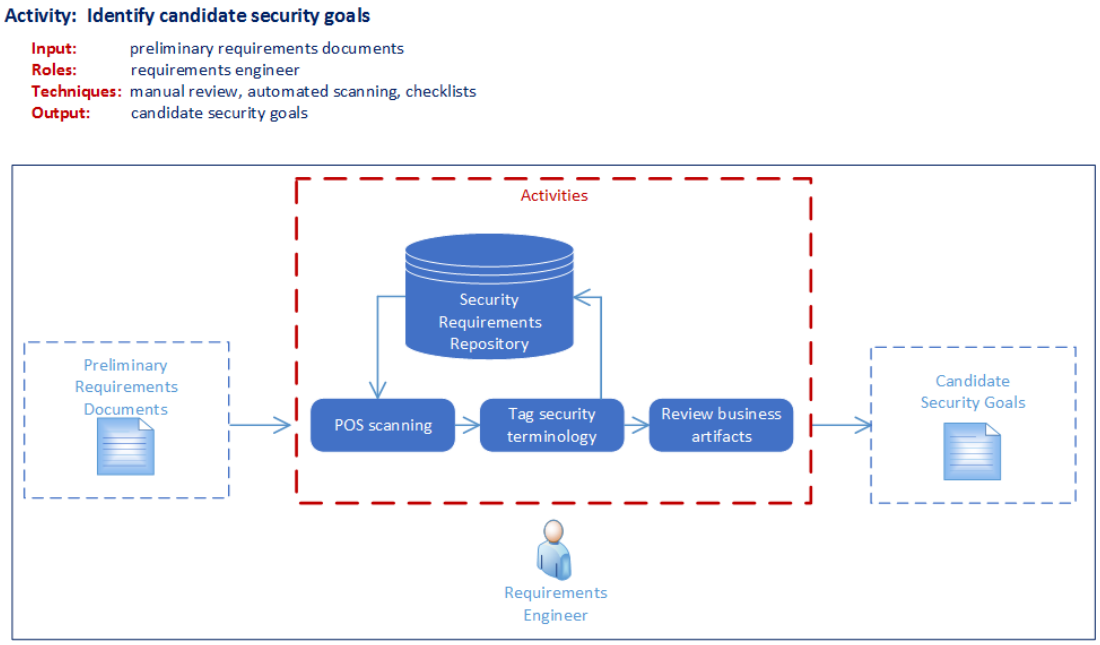

Figure 3.3: Identify Candidate Security Goals Activity

POS Tagging

Online reviews provide seemingly unbiased opinions about products and services that can be used to inform consumers about purchasing decisions. Many web sites have an area for users to post and share comments about products or services. Ratings and opinion comments are the general form of an online review providing a quick snapshot of standing as well as testimonials. As the popularity of online reviews increased, businesses realized the strategic advantages that online reviews present to influencing reputation and purchasing decisions of consumers. A good review can

significantly boost sales whereas as poor review can have a detrimental impact on business. Therefore, extracting or capturing sentiment from online reviews, or opinion mining, has been an active area of recent research (Dave, Lawrence, & Pennock, 2003; Harris, 2012; Hu & Liu, 2004a, 2004b; Jindal & Liu, 2008; Ku, Liang, & Chen, 2006).

Aggregating and extracting meaning from online reviews requires understanding the nature of posted reviews. Online reviews are created when a user posts opinions about products and services. Reviews typically allow the user to choose from a rating, such as a scale of one to ten stars, as well as entering text reviews. User reviews are typically in commentary form where the reviewer can enter opinions in their own words about a product and are intended to enhance the usefulness of a rating and provide additional product insight. Aggregating ratings is not a statistically complicated task, but analyzing text reviews presents a challenge since quantitative analysis methods cannot be easily applied to extract meaning from natural language input. To address this problem, opinions or sentiment must be categorized and extracted from text reviews.

Much of the research in opinion mining is based on data mining and natural language processing techniques. Techniques use a combination of training data, human classification and fully automated processes with varying accuracy and performance results. POS tagging is one proposed method to extract opinions from reviews (Hu & Liu, 2004a; Ku et al., 2006) and is commonly applied to identify features as noun phrases and opinions as close proximity adjectives. Genre

identification and text categorization is one approach used to automate review classification (Ott, Choi, Cardie, & Hancock, 2011). Parsing tools, such as the Stanford Parser, are also available to automate POS tagging and determine word frequency.

Although automated processes were used in many of the approaches, other work relied on human experts manually identifying words or phrases that indicate opinion sentiment (Harris, 2012). For small data sets, manual review is feasible if automated approaches are not available. Larger data sets, such as those requiring crawling millions of reviews, use automated approaches to create smaller data sets that are then reviewed manually. Researchers frequently cite the difficulty building a dataset since it is arduous for human evaluators to manually cull through large numbers of reviews. Therefore, many approaches also include creating a repository of relevant terms that are refined over time. Another goal is to create a "gold standard" dataset that can be used among researchers and refined over time to improve opinion mining results.

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 on the desired end result. Machine learning techniques are desired, but the use of experts can be effective and efficient until these techniques mature and are refined.

Word frequency analysis is commonly used in opinion mining but is not as useful when applied to security requirements extraction. Opinion mining seeks to find similarities among disparate reviewers for a specific product or service. Reviews are typically short in length, informal and are intended to convey a specific message. In contrast, software products are developed for a specific set of stakeholders and there is a single software requirements document that is formal and lengthy. Therefore, counting the frequency of a specific term within a set of reviews reveals different information than the term frequency of a single document. However, if the frequency of security related terms is high in a software requirements artifact, term frequency could indicate then need to define security requirements.

Proximity of terms may reveal relevant information within a software requirements document. For example, 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. Security terms should therefore be tagged and follow-up analysis performed to determine if security requirements can be captured. This is the proposed method in which POS tagging will be implemented to discover security requirements. Additional details on POS tagging are discussed in the identify activity of the security requirements elicitation approach.

One method to identify security terminology implied in requirements is through the use of POS tagging. The requirements engineer takes as input preliminary requirements documents. These documents can be draft software

requirements specifications (SRS), requests for proposals (RFP's), or other requirements specification documents 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.

Candidate security goals (CSG) are general requirements written with implied security needs that may be developed into security requirements. For example, a requirements artifact was scanned and tagged for the word malicious. The following functional requirements (FR) were found:

FR–1:	"Malicious requests are detected and rejected"
FR–2:	"Malicious requests are identified and acted upon"

The requirements engineer would tag the location(s) where the term malicious was found and generate a CSG such as:

CSG-1:	The system shall identify, detect and determine appropriate responses to malicious requests

Further examination of the requirements artifacts also reveals that requests are related to access policies. The CSG can be refined to include this information:

CSG-1:	The system shall identify, detect and determine appropriate responses to malicious requests using access control policies

After all artifacts have been scanned, tagged and reviewed, a candidate security goals artifact will be created as output for the identify activity. This artifact will be used as input to the categorize security goals activity.

3.2.2 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 and business stakeholders work together to review all requirements artifacts that have tagged candidate security goals. Interactive meetings (face-to-face, web facilitated, teleconference) will likely be the most efficient, but virtual document review can also take place. Prior to the meetings, the requirements engineer can assess the goals for quick categorization to facilitate efficient communications. Business stakeholders should be educated on general security principles prior to the meeting. During this activity, each security goal is categorized based on a security principle in order to facilitate additional stakeholder elicitation. Confidentiality, integrity, and availability principles, also referred to as CIA, are the key security principles, but other principles can be defined as well. Each candidate security goal should be categorized with at least one security principle.

Referring to the example CSG from the identify activity; the following security principles can be associated with the CSG:

SP-1:	Confidentiality: protect against unauthorized disclosure of information
SP-2:	Integrity: protect against unauthorized modification or destruction of information

The requirements engineer and business stakeholders will agree upon the general security principles. If a candidate security goal cannot be categorized, additional elicitation and analysis can be iteratively undertaken with the stakeholders. If CSG's still cannot be categorized after additional iterations, it will fail the activity and the CSG will be discarded. The details for the categorize security goals based on security principle activity are shown in Figure 3.4.

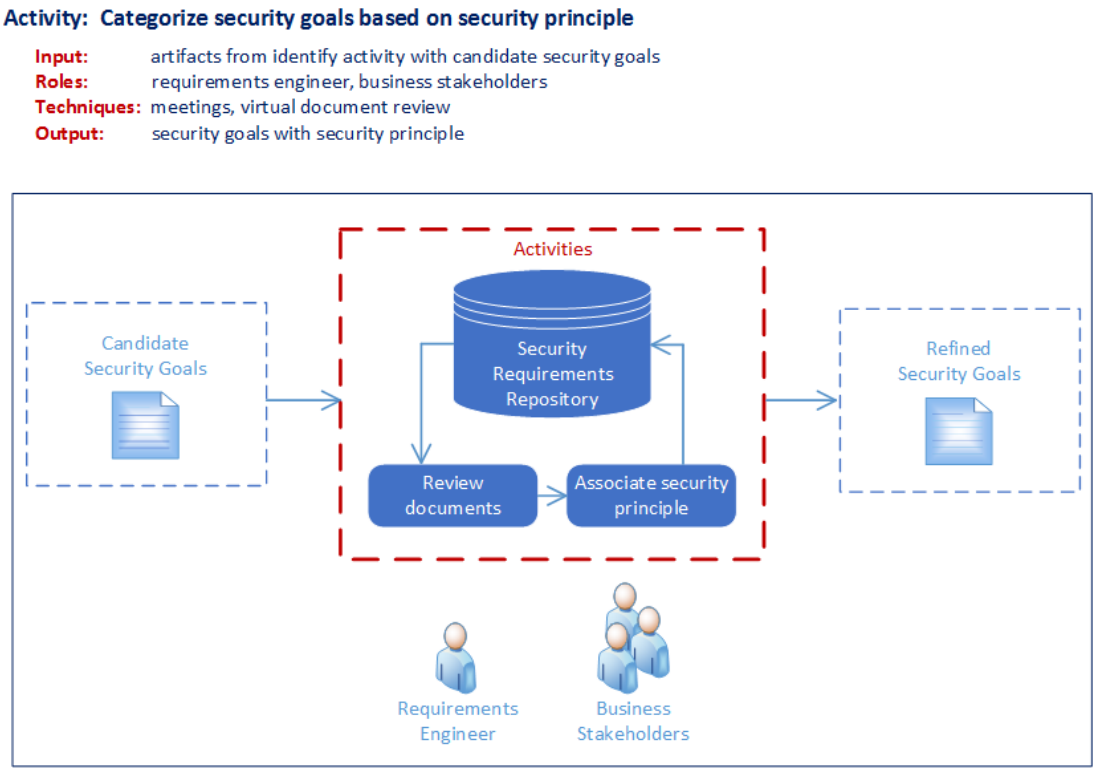

Figure 3.4**:** Categorize Security Goals Activity

3.2.3 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. 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. The output from this activity is a set of preliminary security requirements based on

the CSG's. Continuing with the previous example, the preliminary security requirement (PSR) generated from CSG-1 could be:

PSR-1:	The system shall protect the confidentiality and integrity of data by identifying, detecting and rejecting malicious requests using access control policies

The details for the understand stakeholder goals and develop preliminary security requirements activity are shown in Figure 3.5.

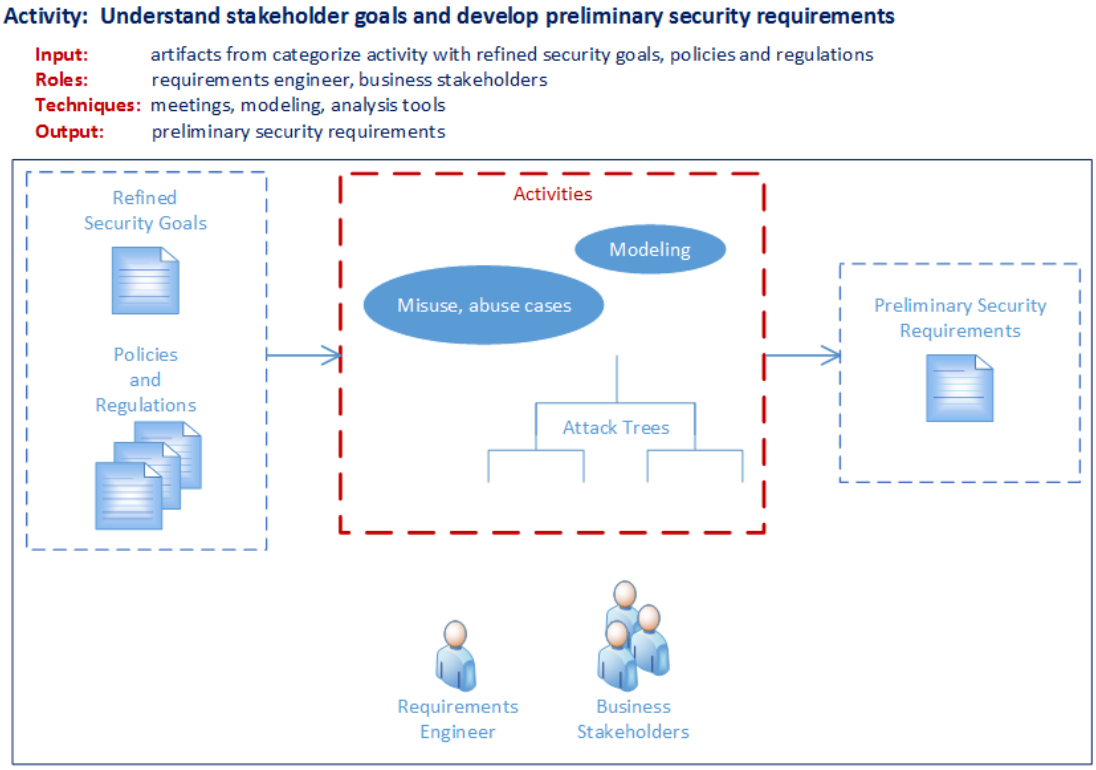

Figure 3.5: Understand Stakeholder Goals Activity

3.2.4 Prioritize Preliminary Security Requirements

Preliminary security requirements need to be prioritized to generate the final security requirements. During this activity, the requirements engineer continues to work with

business stakeholders to analyze the input preliminary security requirements. Recommended analysis techniques are risk management tools commonly used by the stakeholders who will foster familiarity with the process. An additional technique, Failure Modes and Effects Analysis (FMEA), is also recommended.

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.

Analysis requires creating severity, occurrence and detection rankings in order to determine a risk priority number (RPN). A standard scale for severity, occurrence and detection can be adopted similar to Table 3.3 as a starting point for FMEA analysis. Experienced FMEA users may develop more refined rankings similar to Table 3.4, Table 3.5, and Table 3.6. The RPN is calculated as the product of the risk rankings.

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

Continuing from the previous activities, Table 3.7 demonstrates analyzing the preliminary security requirement related to malicious requests (SR-1). 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. Effects related to loss of confidentiality and integrity are determined to be viewed, stolen or corrupted data. Rankings 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 by a malicious request significantly outweighs other effects. Using the FMEA results, requirements engineer and business stakeholders will refine the preliminary security requirements until a list of final security requirements has been generated. The details for the prioritize security requirements activity are shown in Figure 3.6.

Table 3.3: FMEA Standard Scale

Standard Scale for Severity, Occurrence or Detection

Impact	Rating	Criteria: A Failure Could…
Very High	**9-10**	virtually inevitable
High	**8-7**	failure likely, many known cases
Moderate	**4-6**	somewhat likely, some known cases
Low	**3-2**	few known cases
Unlikely	**1**	no known cases

Table 3.4: FMEA Severity Scale

Severity Scale = Likely Impact of Failure

Impact	Rating	Criteria: A Failure Could…
Bad	**10**	Injure a customer or employee
	9	Be illegal
	8	Render the software unfit for use
	7	Cause extreme customer dissatisfaction
	6	Result in partial malfunction
	5	Cause a loss of performance likely to result in a complaining
	4	Cause minor performance loss
	3	Cause a minor nuisance; can be overcome with no loss
	2	Be unnoticed; minor effect on performance
Good	**1**	Be unnoticed and not affect the performance

Table 3.5: FMEA Occurrence Scale

Occurrence Scale = Frequency of Failure

Impact	Rating	Time period	Probability of Occurrence
Bad	**10**	More than once per day	> 30%
	9	Once every 3-4 days	≤ 30%
	8	Once per week	≤ 5%
	7	Once per month	≤ 1%
	6	Once every 3 months	≤ 0.3 per 1,000
	5	Once every 6 months	≤ 1 per 10,000
	4	Once per year	≤ 6 per 100,000
	3	Once every 1-3 years	≤ 6 per million (approx. six sigma)
	2	Once every 3-6 years	≤ 3 per 10 million
Good	**1**	Once every 6-100 years	≤ 2 per billion

Table 3.6: FMEA Detection Scale

Detection Scale = Ability to Detect Failure

Impact	Rating	Definition
Bad	**10**	Defect caused by failure is not detectable
	9	Occasional units are checked for defects
	8	Units are systematically sampled and inspected
	7	All units are manually inspected
	6	Manual inspection with mistake proofing modifications
	5	Process is monitored with control charts and manually inspected
	4	Control charts used with an immediate reaction to out-of-control condition
	3	Control charts used as above with 100% inspection surrounding out-of-control condition
	2	All units automatically inspected or control charts used to improve the process
Good	**1**	Defect is obvious and can be kept from the customer or control charts are used for process improvement to yield a no-inspection system with routing monitoring

Table 3.7: FMEA Analysis of Security Requirements

Failure	Effect	Severity	Occurrence	Detection	RPN
malicious request	data viewed	3	7	9	189
malicious request	data stolen	9	4	9	324
malicious request	data corrupted	5	4	4	80

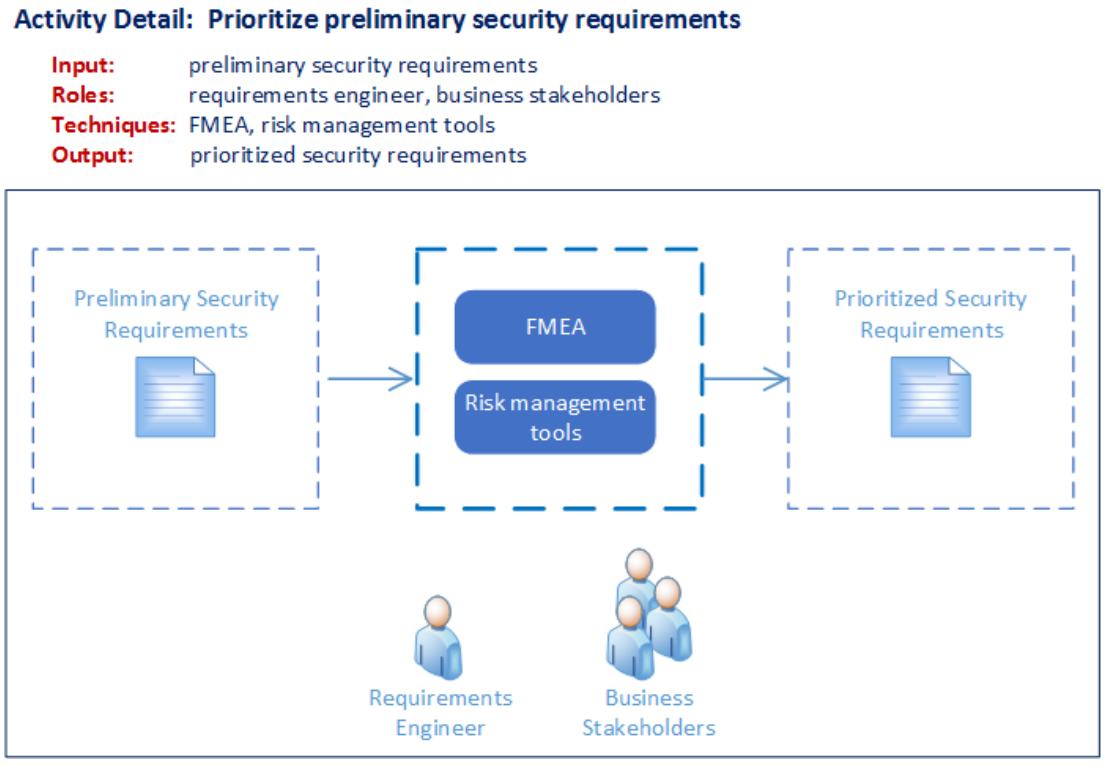

Figure 3.6: Prioritize Preliminary Security Requirements

4 Research Results Evaluation and Validation

The security requirements elicitation approach will be evaluated empirically by analyzing publically available software requirements specifications (SRS). An internet search of pdf and Word documents was conducted using the search term "software requirements specification". Many student project SRS documents are available from .edu sites and these were filtered out from the search. Template documents were also discarded. A base set of 46 SRS documents were downloaded of which three contained sections specifically for security requirements. The remaining 43 SRS documents were used analyzed using POS tagging. After tagging analysis, a smaller subset of the tagged documents was selected and analyzed using the security requirements elicitation steps. We present POS tagging, security requirements elicitation and results next.

4.1 POS Tagging

A set of security terminology was required in order to scan documents and conduct POS tagging. The security terms chosen were based on manually reviewing SRS documents containing security requirements and the author's knowledge. The resulting set of security terms is show in Table 4.1. Unigrams were chosen for scanning rather than n-grams (short security phrases). Similar terms such as authenticate and authentication as well as plural forms of some terms were included

due to the requirements of the POS tagging software. The set of security terms would be refined and updated after preliminary results are evaluated for future iterations of the approach. Analysis of term frequency, false positives and term relevancy will be used to prune or expand the security term dataset.

We developed a POS scanner to scan and tag the set of SRS documents. Small organizations are likely to generate SRS documents using word processing software rather than sophisticated software development management software. All pdf documents were converted to Word 2010 format (.doc) in preparation for scanning. The scanning software was written in Visual Basic for Applications (VBA) which integrates with Microsoft Word and can easily facilitate the scanning process. The basic steps in the scanning process are:

1. Open the document
2. Clear all bookmarks
3. Scan for, count and bookmark the location of each security term
4. Write the document name, security term and frequency to a text file
5. Save and close the document

Multiple files can be automatically scanned sequentially. The entire scanning and tagging process is automated and processing time was approximately 1.5 minutes per document.

Table 4.1: Security Terminology Frequency and Rank

Security Terminology		
Security Term	**Frequency**	**Rank**
access	416	2
audit	28	10
authenticate	5	17
authentication	30	8
authorize	0	19
authorized	146	5
certificate	205	4
certificates	85	7
deny	3	18
encrypt	12	14
encryption	20	12
https	14	13
logon	8	15
malicious	8	15
password	237	3
permission	86	6
privileges	24	11
risk	30	8
security	551	1
Number of documents scanned: 43		

Table 4.2: SRS Document Security Term Frequency

Security Term Frequency per SRS Document					
Doc #	**Frequency**	**Doc #**	**Frequency**	**Doc #**	**Frequency**
1	119	16	20	31	54
2	24	17	20	32	52
3	20	18	41	33	63
4	14	19	113	34	56
5	35	20	27	35	36
6	22	21	83	36	52
7	701	22	141	37	50
8	17	23	90	38	64
9	44	24	31	39	84
10	29	25	183	40	73
11	36	26	26	41	43
12	21	27	35	42	47
13	18	28	87	43	49
14	14	29	44		
15	27	30	49		
Average Frequency of Security Terms per Document: 66.4					
Total Security Term Frequency: 2854					
Total SRS Documents Scanned: 43					

4.1.1 Analysis of Tagged Security Terms

Table 4.1 lists the security term frequency and relative ranking. Five security terms with the highest frequency are security, access, password, certificate, and authorized. Security terms with the lowest frequency are authorize, deny, authenticate, logon and malicious. Figure 4.1 and Figure 4.2 graphically display the security term frequency and average frequency for each of the selected security terms. Table 4.2 shows the per document tagging statistics. The security term frequency per document revealed a total of 2,854 terms tagged with an average per document frequency of 66.4. Tagged term frequency ranged from a low of 14 to a high of 701. The average term frequency may be skewed by one document that has a very high term frequency. Without this document the average is closer to 51 but even at 66.4, it is low enough that manual review by a requirements engineer would not be cumbersome. We will analyze the SRS documents to determine if the size of the security term dataset impacts the viability of discovering candidate security goals. Results from the elicitation activities will be analyzed to determine if the set of security terms can be pruned to a smaller set or if additional security terms are needed to generate security requirements.

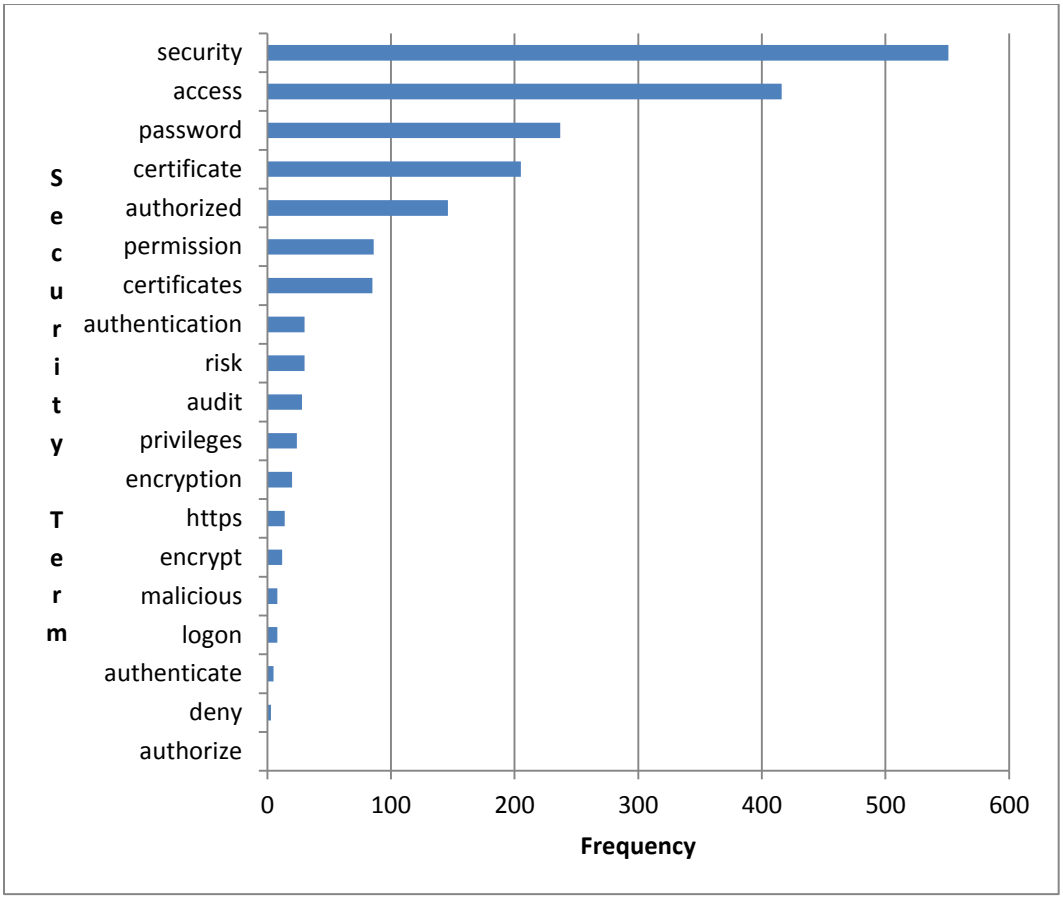

Figure 4.1: Security Term Frequency from POS Tagging

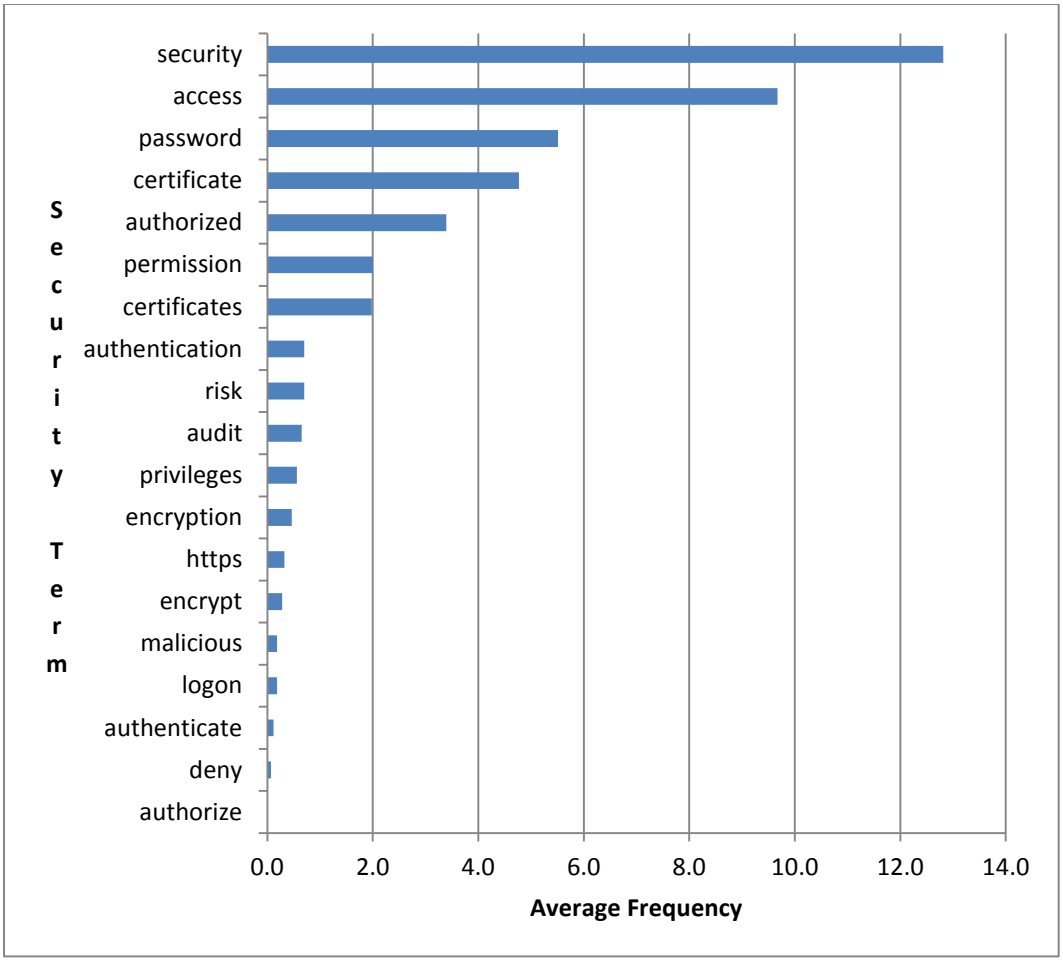

Figure 4.2: Security Term Average Frequency from POS Tagging

4.2 Security Requirements Elicitation

Eight tagged documents with the highest frequency were chosen for further analysis using the security requirements elicitation activities. One of the documents was eliminated due to formatting issues. The document with the highest security term frequency was a highly complex document that specified multiple sub-systems and contained hundreds of functional requirements. The complexity of the SRS was not

representative of the type of product that would be developed by a small organization and was also eliminated.

4.2.1 Identify Candidate Security Goals

Each document was manually reviewed to determine if the tagged security terms were relevant to identifying candidate security goals. Custom code facilitated the process by selecting each tagged term and allowing the reviewer to accept or reject each term based on the context of the language surrounding each term. Terms could be rejected (false positives) for a variety of reasons. Acronym lists, glossaries and references to other documents were common reasons for rejecting or "un-tagging" a term. Other terms were found to have a different meaning such as "certificate" paired with nouns that are not related to security such as "ship certificate". "Access" was another term that was frequently paired with "channel" in another document. Other terms were repeated in close proximity, typically separated by a few words or in a nearby sentence. When identical close proximity terms were found, only one of the terms remained tagged. On the average, 15% of the tagged terms remained for an average of 27 terms per SRS document. Figure 4.3 and Figure 4.4 show the results of security term frequency before and after false positives are removed.

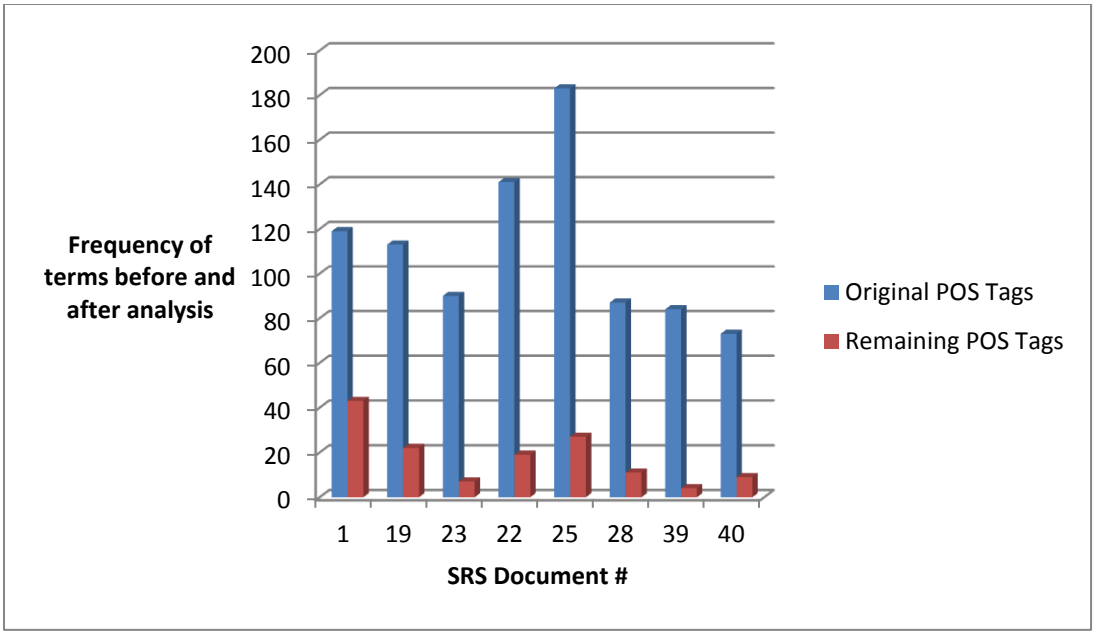

Figure 4.3: Comparison of Original and Remaining Term Frequency

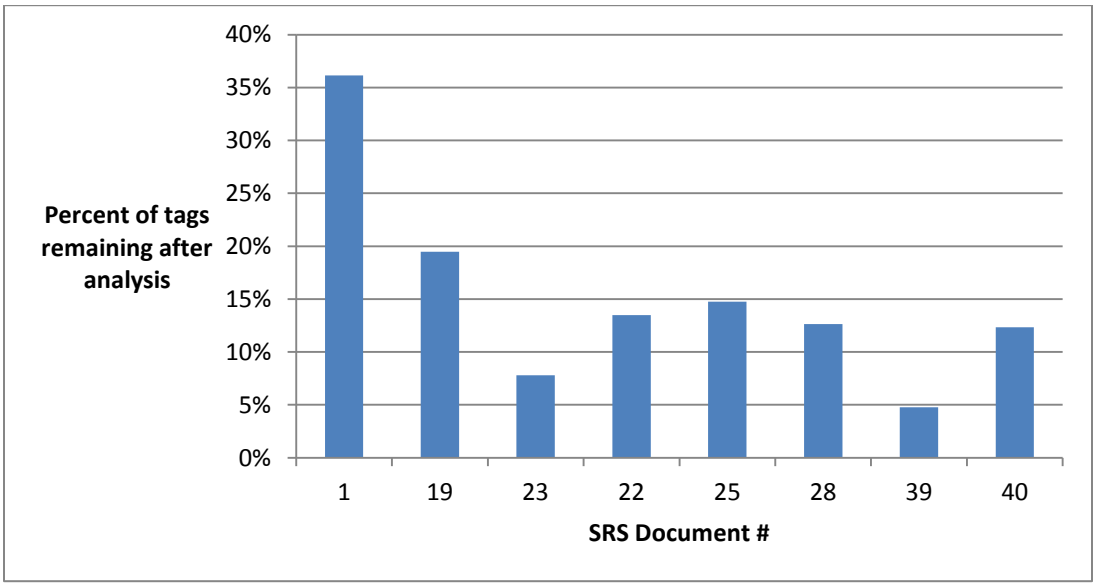

Figure 4.4: Average Security Term Frequency After Reduction

Carrying out the identify activity requires that the remaining security terms are analyzed to determine to identify candidate security goals (CSG). Analysis from one of the SRS documents reveals the following CSG's:

CSG-1:	The application will also allow for remote access through a firewall via outside telecommunications networks by authorized users.
CSG-2:	The logon screen shall request user name and corresponding password.
CSG-3:	For system login purposes, the hash function shall also be used to encrypt user passwords.

4.2.2 Categorize Security Goals Based On Security Principle

Each of the CSG's is categorized based on security principle. Security principles (SP) are commonly known as the CIA triad which stands for confidentiality, integrity and availability. Common definitions for the security principles are:

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

The CSG's can be categorized with multiple security principles. If no security principles can be applied, the CSG would be rejected.

CSG-1:	SP-1, SP-2
CSG-2:	SP-2
CSG-3:	SP-2

4.2.3 Understand Stakeholder Goals and Develop Preliminary Security Requirements

Stakeholder goals are elicited for each of the categorized CSG's and preliminary security requirements (PRS) are developed.

PSR-1:	The system shall protect confidentiality and integrity of data by allowing remote access through a firewall ...only to authorized users.
PSR-2:	The system shall protect integrity of data by requesting a user name and password prior to access.
PSR-3:	The system shall protect confidentiality of user passwords by encrypting passwords.

4.2.4 Prioritize Preliminary Security Requirements

FMEA analysis is performed on for each PSR. Potential failure modes and effects are identified. The failure modes and effects are written in general terms for ease of understanding and quick analysis. Severity, occurrence, and detection ratings were assigned using the standard FMEA scale shown in Table 3.3. The FMEA analysis is shown in Table 4.3. If the RPN is determined to be above a minimum threshold, the PSR will be accepted. Security requirements that are accepted will be included in the

SRS as security requirements and will be subject to additional modeling and validation activities included in later software development activities. The activities in the security requirements elicitation approach are documented using the template shown in Table 4.4. All of the candidate security requirements were previously identified as general security requirements. FMEA analysis confirms the need for security requirements. The refined requirements can now be accepted as security requirements and can be input into the security requirements repository.

Table 4.3: FMEA Analysis of Preliminary Security Requirements

Failure	Effect	Severity	Occurrence	Detection	RPN
remote access by unauthorized user	data viewed	4	3	7	84
remote access by unauthorized user	data stolen	7	3	9	189
remote access by unauthorized user	data corrupted	5	3	7	105
access by unauthorized user	data viewed	4	3	7	84
password compromised	data viewed	6	3	7	126
password compromised	data stolen	6	3	9	162
password compromised	data corrupted	5	3	7	105

Table 4.4: Security Requirements Elicitation Template

Security Requirements Elicitation			
Document Name:			
Document ID:	19	Original tag count	113
Project ID:		Final tag count	43
1 Identify candidate security goals			
Candidate Security Goals (CSG)			
CSG - 1	The application will also allow for remote access through a firewall via outside telecommunications networks by authorized users.		
CSG - 2	The logon screen shall request user name and corresponding password.		
CSG - 3	For system login purposes, the hash function will also be used to encrypt user passwords.		
2 Categorize security goals based on security principle			
Apply security principle(s) to CSG			
CSG - 1	SP-1, SP-2		
CSG - 2	SP-2		
CSG - 3	SP-2		
3 Understand stakeholder goals and develop preliminary security requirements			
Preliminary Security Requirement (PSR)			
PRS - 1	The system shall protect confidentiality and integrity of data by allowing remote access through a firewall ...only to authorized users.		
PRS - 2	The system shall protect integrity of data by requesting a user name and password prior to access.		
PRS - 3	The system shall protect confidentiality of user passwords by encrypting passwords.		
4 Prioritize preliminary security requirements			
PSR	**Effect**	**FMEA RPN**	**Accept/Reject**
PRS - 1	Data Stolen	189	Accept
PRS - 2	Data viewed	84	Accept
PRS - 3	Password compromised	162	Accept
Prioritized Security Requirements (SR)			
SR – 1	The system shall protect confidentiality and integrity of data by allowing remote access through a firewall ...only to authorized users.		
SR – 2	The system shall protect integrity of data by requesting a user name and password prior to access.		
SR – 3	The system shall protect confidentiality of user passwords by encrypting passwords.		
Notes			
All of the identified requirements should be reclassified as security requirements.			

4.3 **Analysis of Security Requirements Elicitation Approach**

The remaining documents were analyzed to determine if security requirements could be elicited using POS tagging and the security requirement elicitation approach. Table 4.5 is the analysis of the SRS documents with the highest term frequency. Security requirements specified indicates if a specific subsection of security requirements was included in the SRS. If any of the identified security requirements were originally identified as functional or non-functional requirements and were subsequently determined to be security requirements, convert to security requirements is marked as "Yes". The number of identified security requirements from the analysis is also indicated. Finally, general comments from the analysis are included.

Table 4.5: Security Requirements Analysis of SRS Documents

Document #:	1
Security Requirements Specified:	No
Convert to Security Requirements:	Yes
Identified Security Requirements:	9
Contained use cases that discussed security concepts which could be converted to abuse or misuse cases. Several functional requirements were in fact security requirements that should be re-written and classified as security requirements. Non-functional requirements were generally used to address security requirements (verifiability and security). The verifiability section included requirements that addressed security concerns (terms used were suspicious records, authorized user, audits, special privileges). All of these concerns should be converted to security requirements. Security section described specific security mechanisms. All of the identified security requirements had at least one tagged security term.	

Document #:	19
Security Requirements Specified:	No
Convert to Security Requirements:	Yes
Identified Security Requirements:	12

Security requirements were scattered within other requirements (external system interfaces, communications interfaces, functional requirements). One interesting note was found regarding system login (hash function shall be used to encrypt passwords, question in document asking why this was a requirement).

Document #:	22
Security Requirements Specified:	No
Convert to Security Requirements:	Yes
Identified Security Requirements:	6

Very ambiguous and general requirements. Authentication in the form of password requirements were the key security requirements. Categorizing by security principles was effective since most of the tagged terms eluded to data integrity and confidentiality principles. Most of the security requirements can be developed from existing functional and non-functional requirements.

Document #:	23
Security Requirements Specified:	No
Convert to Security Requirements:	No
Identified Security Requirements:	4

Contained use cases to define the functional requirements. Non-functional requirements contained a section on security that had a mash-up of policy and training information. Definitions of strong passwords practices (length, special characters, numbers, password reuse) were included. The basic specification was to require authentication using strong passwords or digital certificates. There were only mild requirements that could be converted to stronger security requirements and many new security requirements could be added to account for data confidentiality and integrity.

Document #:	25
Security Requirements Specified:	No
Convert to Security Requirements:	Yes
Identified Security Requirements:	8
"Access" security term generally defined access controls. Data confidentiality and integrity are also key considerations. Comments were of a form such as: "There should be security preventing the intrusion into the system by unauthorized users, or users at unauthorized access levels."	
Document #:	28
Security Requirements Specified:	Yes
Convert to Security Requirements:	Yes
Identified Security Requirements:	6
There is a security requirements section, but it is only a paragraph that broadly identifies security characteristics. Encryption, passwords, access controls (data integrity and confidentiality) are all specifically addressed. Several security requirements were contained within functional requirements.	
Document #:	39
Security Requirements Specified:	No
Convert to Security Requirements:	Yes
Identified Security Requirements:	4
Security requirements were contained within functional requirements. Permission levels, access controls, non-repudiation, encryption, and passwords were commonly used terms. Use cases were identified that could be modified into misuse or security use cases.	
Document #:	40
Security Requirements Specified:	No
Convert to Security Requirements:	Yes
Identified Security Requirements:	6
Contained use cases. Security primarily for passwords and access with references to "secure connections". Contained references to legislation and regulations.	

POS tagging revealed interesting data related to the relevancy of security terms when identifying security requirements. Security terms with the highest frequency from initial scanning had a lower retention rate after manual review and pruning of security terms. Terms such as “password” and “authentication” were heavily used and often repeated within close proximity. Security requirements were frequently developed when a high concentration of security terms within a sentence or in neighboring sentences was found Lower frequency terms were often not located in close proximity to other security terms but did identify security requirements. These requirements were more subtly implied and would likely require additional elicitation and modeling with business stakeholders to fully understand the security goals. Figure 4.5 and Figure 4.6 illustrate the analysis of security term frequency after an initial review of tagged security terms was conducted.

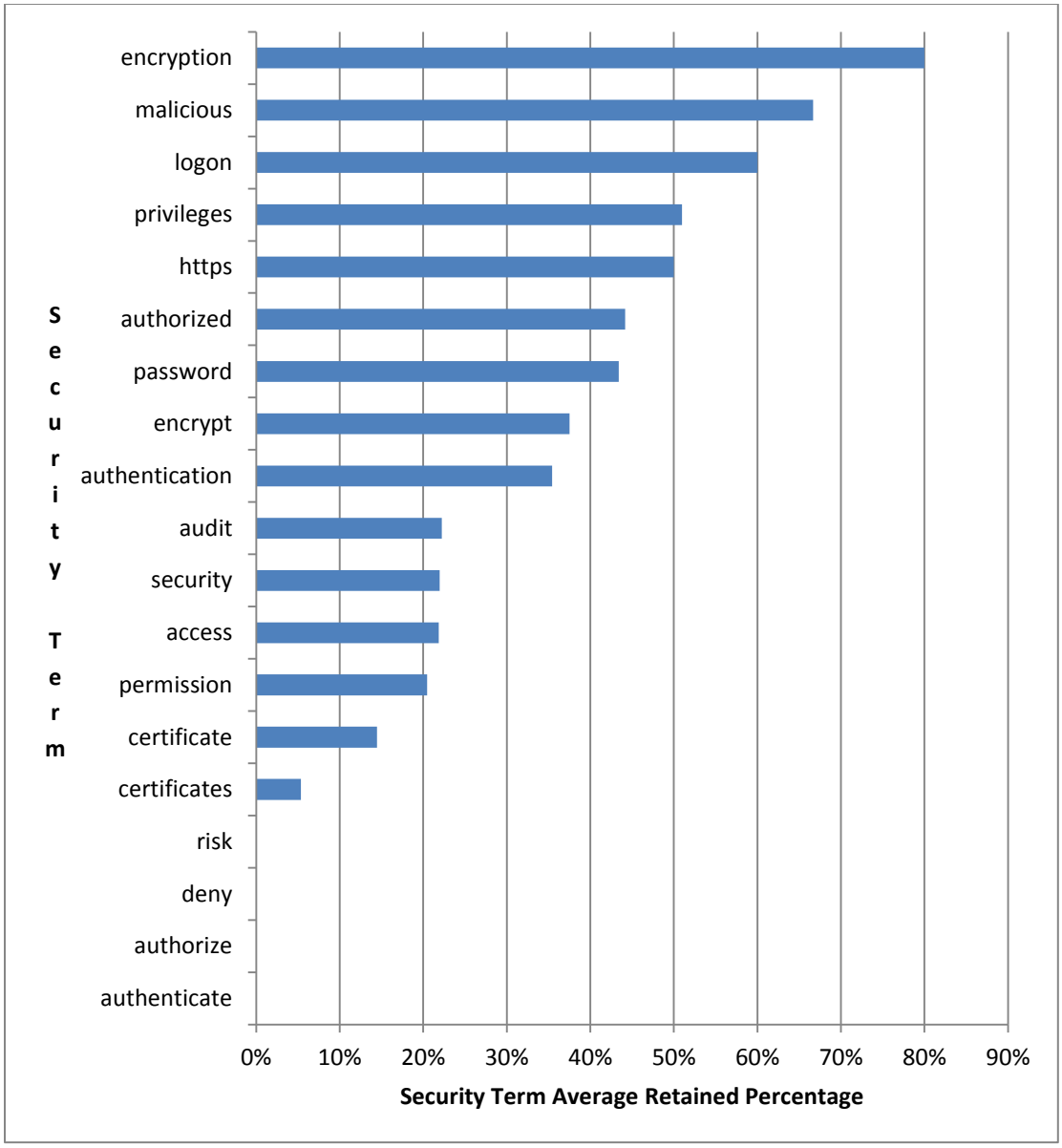

Figure 4.5: Percentage of Security Terms Retained After Initial Review

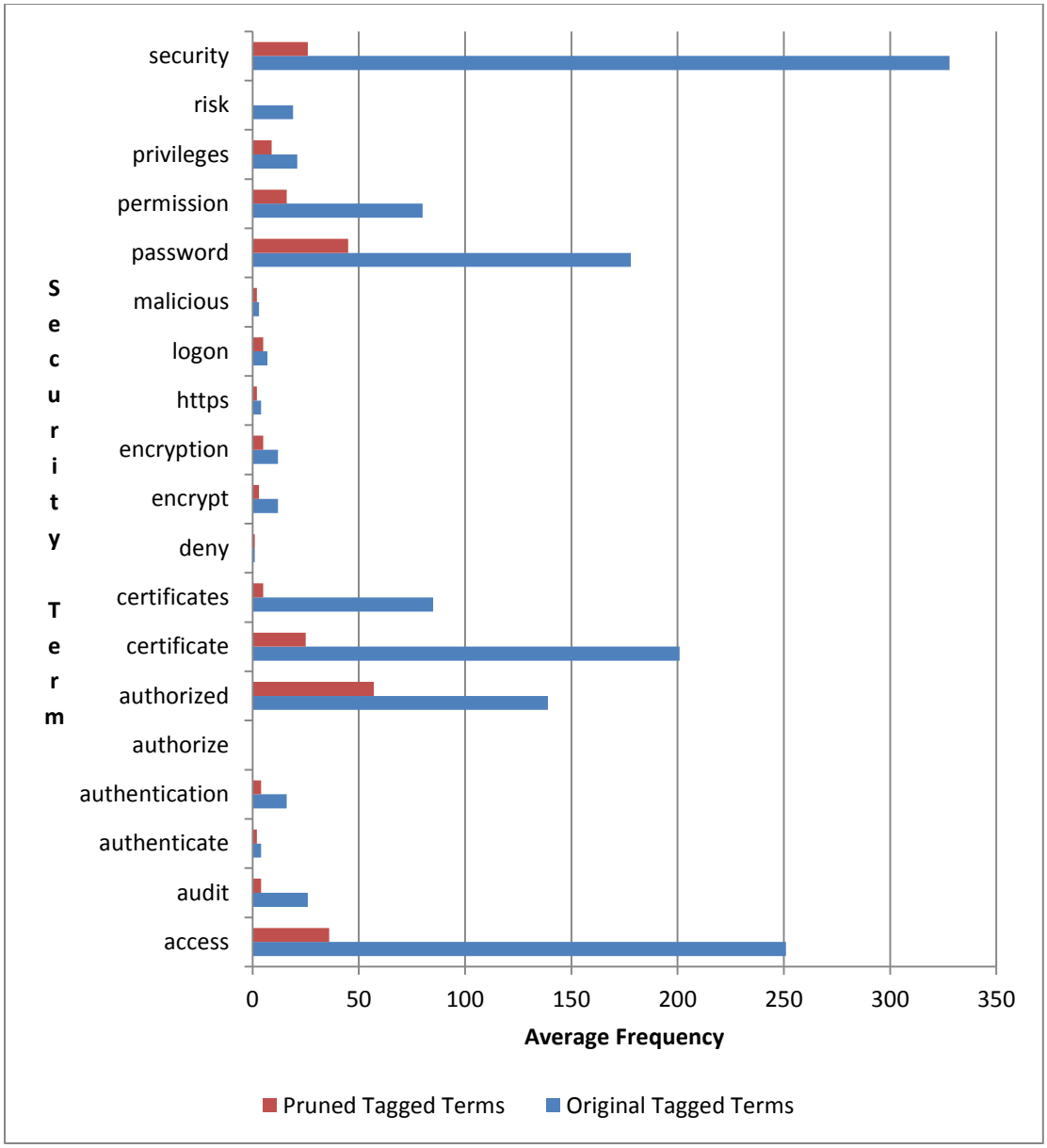

Figure 4.6: Comparison of Original and Pruned Security Term Frequency

4.4 Feasibility of the Proposed Solution

Feasibility of the proposed solution should be taken into consideration given that we are targeting small, agile organizations. Drawbacks to the approaches discussed in the survey chapter included approach complexity, resources and security expertise. Given a basic set of security terms and the scanning software, POS tagging can be accomplished with minimal time and personnel resources. Pruning the tagged terms is a manual task, but does not require advanced security expertise and can be accomplished in a relatively short time. The 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 the 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 are desired. The development of a security requirements repository to improve traceability and reusability as the elicitation approach matures does not detract from the feasibility of the proposed solution. Therefore, the proposed security requirements elicitation approach is a feasible alternative to other approaches for small, agile organizations.

4.5 Summary

This chapter presented an analysis of POS tagging and the security requirements approach. Sample SRS documents were collected and automated scanning software developed for this thesis was used for POS tagging of security terms. A subset of SRS documents with the highest frequency of tagged security terms was analyzed. Tagged terms that were redundant or were false positives had tags removed. Next, the activities in the security requirements elicitation approach were undertaken. Security requirements that had not been previously identified were elicited and developed from all of the SRS documents.

5 Conclusions and Future Work

5.1 Summary

This thesis describes a solution for eliciting security requirements using POS tagging which can be implemented by small, agile organizations. Resulting security requirements are integrated into SRS documents and a security requirements repository enables rapid reuse of developed requirements. Key elements of the elicitation solution are (1) identifying security goals, (2) categorizing goals by security principle, (3) understanding stakeholder goals to develop 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 solution is flexible in order to accommodate the needs of small, agile software development organizations but outlines a basic structure that can be easily implemented. The solution takes place at the earliest phase of the software development process during requirements elicitation in order to reduce cost and rework at later stages of development.

A POS scanning algorithm was developed as part of the solution to automate early discovery of security goals by tagging security terms within a document. The scanning algorithm can be used with individual documents or to scan multiple documents at one time. 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 proposed solution. Security requirements were discovered and refined in documents obtained from a sample set of publically available SRS documents. These results verify that solution is feasible and can be implemented in small, agile organizations.

5.2 Research Contributions

Our research provides two major contributions. The first contribution is the POS scanning algorithm. We use POS tagging to discover security requirements from existing requirements artifacts by extracting implied security goals from business stakeholders. POS tagging jump starts the elicitation process and focuses efforts on specific areas of the requirements document for further examination. This approach differs from surveyed works that either relies on developing complex security models or implementing comprehensive security initiatives. Requirements engineers can have a wide range of security knowledge and expertise to implement the solution rather than needing to be security experts. For small organizations with limited resources, this addresses their needs to build security maturity over time rather than undertaking comprehensive security initiatives. By starting with a basic set of security terms and understanding of key security principles, POS tagging focuses resources on fully understanding security goals. The second contribution is development of a four step process to elicit, analyze, prioritize and document security requirements. A key component of prioritization is the implementation of FMEA

analysis which has roots in Six Sigma methodologies. FMEA analysis has not previously be considered as an approach that can be used as part of the requirements elicitation, but has advantages in that it is quick, easy to understand by non-technical stakeholders and aids in prioritization of security requirements. RPN results are based on ranking risk based on frequency, occurrence and detection each of which can be addressed individually to reduce risk. The solution is flexible and the scope of effort can be adjusted to accommodate resources available for a software project.

Previous works focused on modeling scenarios, addressing specific threats, implementing security mechanisms, or developing broad security initiatives. However, if stakeholders do no clearly understanding security needs, deriving security requirements using these approaches can be a difficult and resource intensive exercise. The security requirements elicitation solution is designed to be integrated into the requirements elicitation phase of software development in order to reduce costly rework at later stages of development.

5.3 Suggestions for Future Work

The focus of this work has been the integration of POS tagging within a security requirements elicitation approach. During evaluation of the solution, we observed that additional work in POS tagging is needed. 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. Furthermore, the development of a security requirements language using a formal language such as Backus Normal Form (BNF) notation to precisely and formally define security requirements and generate a reusable repository of security requirements. Finally, failure modes and effects analysis could be used to generate techniques, such as abuser stories, which are commonly used with agile development elicitation and modeling techniques.

Bibliography

Alexander, I. (2003). Misuse cases help to elicit non-functional requirements. *Computing & Control Engineering Journal, 14*(1), 40-45.

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

Barnum, S., & Sethi, A. (2006). Introduction to attack patterns Retrieved 01/27/2011, from `https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/attack/585-BSI.html`

Boehm, B. (2002). Get ready for agile methods, with care. *Computer, 35*(1), 64-69.

Boström, G., Wäyrynen, J., Bodén, M., Beznosov, K., & Kruchten, P. (2006). *Extending XP practices to support security requirements engineering*. Paper presented at the Proceedings of the 2006 international workshop on Software engineering for secure systems, Shanghai, China.

CERT-SEI. (2010). SQUARE Tool, from `http://www.cert.org/sse/square/square-tool.html`

Dave, K., Lawrence, S., & Pennock, D. M. (2003). *Mining the peanut gallery: opinion extraction and semantic classification of product reviews*. Paper presented at the Proceedings of the 12th international conference on World Wide Web, Budapest, Hungary.

Devanbu, P. T., & Stubblebine, S. (2000). *Software engineering for security: a roadmap*. Paper presented at the Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland.

Falcarin, P., & Morisio, M. (2004). *Developing Secure Software and Systems.* Paper presented at the IEC Network Security: Technology Advances, Strategies, and Change Drivers, Chicago: International Engineering Consortium (IEC).

Firesmith, D. (2003). Security Use Cases. *Journal of Object Technology, 2*(3), 53-64.

Giorgini, P., Massacci, F., Mylopoulos, J., & Zannone, N. (2004). Requirements engineering meets trust management: model, methodology, and reasoning: University of Trento, Department of Information and Communication Technology.

Giorgini, P., Massacci, F., Mylopoulos, J., & Zannone, N. (2005, 29 Aug.-2 Sept. 2005). *Modeling security requirements through ownership, permission and*

delegation. Paper presented at the Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on.

Gitlow, H. S., & Levine, D. M. (2005). Six Sigma for green belts and champions: Upper Saddle River, NJ: Financial Times Prentice Hall.

Haley, C. B., Laney, R., Moffett, J. D., & Nuseibeh, B. (2008). Security Requirements Engineering: A Framework for Representation and Analysis. *Software Engineering, IEEE Transactions on, 34*(1), 133-153.

Hallberg, N., & Hallberg, J. (2006, 21-23 June 2006). *The Usage-Centric Security Requirements Engineering (USeR) Method.* Paper presented at the Information Assurance Workshop, 2006 IEEE.

Harris, C. G. (2012, July, 2012). *Detecting deceptive opinion spam using human computation.* Paper presented at the Proceedings of the 4th Human Computation Workshop (HCOMP'12), Toronto, Canada.

Hu, M., & Liu, B. (2004a). *Mining and summarizing customer reviews.* Paper presented at the Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining.

Hu, M., & Liu, B. (2004b). *Mining opinion features in customer reviews.* Paper presented at the Proceedings of the National Conference on Artificial Intelligence.

Ingoldsby, T. R. (2009). Attack Tree-based Threat Risk Analysis, from http://www.amenaza.com/downloads/docs/AttackTreeThreatRiskAnalysis.pdf

Jewell, M. (2007). T.J. Maxx theft believed largest hack ever Retrieved 03/20/13, from http://www.nbcnews.com/id/17871485/#.UUnroxxQG5I

Jindal, N., & Liu, B. (2008). *Opinion spam and analysis*. Paper presented at the Proceedings of the international conference on Web search and web data mining, Palo Alto, California, USA.

Jürgens, J. (2001). *Towards Development of Secure Systems Using UMLsec*. Paper presented at the Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering.

Jürgens, J. (2002). *UMLsec: Extending UML for Secure Systems Development*. Paper presented at the Proceedings of the 5th International Conference on The Unified Modeling Language.

Ku, L. W., Liang, Y. T., & Chen, H. H. (2006). *Opinion extraction, summarization and tracking in news and blog corpora.* Paper presented at the Proceedings of AAAI-2006 Spring Symposium on Computational Approaches to Analyzing Weblogs.

Lodderstedt, T., Basin, D. A., & Doser, J. (2002). *SecureUML: A UML-Based Modeling Language for Model-Driven Security*. Paper presented at the Proceedings of the 5th International Conference on The Unified Modeling Language.

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

Malone, B., & Siraj, A. (2008). *Tracking requirements and threats for secure software development*. Paper presented at the Proceedings of the 46th Annual Southeast Regional Conference on XX, Auburn, Alabama.

McGraw, G. (2005). 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, 13*(9), 42-43.

McGraw, G. (2008). Automated Code Review Tools for Security. *Computer, 41*(12), 108-111.

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

Mead, N. R., Hough, E., & Stehney II, T. (2005). Security Quality Requirements Engineering. *Technical Report CMU/SEI-2005-TR-009*, from `http://www.sei.cmu.edu/library/abstracts/reports/05tr009.cfm`

Mead, N. R., & Stehney, T. (2005). *Security quality requirements engineering (SQUARE) methodology*. Paper presented at the Proceedings of the 2005 workshop on Software engineering for secure systems\—building trustworthy applications, St. Louis, Missouri.

Mercuri, R. T., & Neumann, P. G. (2003). Security by obscurity. *Communications of the ACM, 46*(11), 160.

Moffett, J. D., Haley, C. B., & Nuseibeh, B. (2004). Core security requirements artefacts. *Department of Computing, The Open University, Milton Keynes, UK, Technical Report, 23*.

Ott, M., Choi, Y., Cardie, C., & Hancock, J. T. (2011). Finding deceptive opinion spam by any stretch of the imagination. *arXiv preprint arXiv:1107.4557*.

Peeters, J. (2005). *Agile Security Requirements Engineering.* Paper presented at the Symposium on Requirements Engineering for Information Security 2005.

Romero-Mariona, J. (2009). *Secure and Usable Requirements Engineering*. Paper presented at the Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering.

Savolainen, J., Kuusela, J., & Vilavaara, A. (2010). *Transition to agile development-rediscovery of important requirements engineering practices.* Paper presented at the Requirements Engineering Conference (RE), 2010 18th IEEE International.

Tondel, I. A., Jaatun, M. G., & Meland, P. H. (2008). Security Requirements for the Rest of Us: A Survey. *Software, IEEE, 25*(1), 20-27. doi: 10.1109/ms.2008.19

Viega, J. (2005). Security - Problem Solved? *Queue, 3*(5), 40-50. doi: http://doi.acm.org/10.1145/1071713.1071728

Appendix A

Characterizing a Small, Agile Organization

A software development company's requirements engineering processes were studied to determine representative characteristics for a small, agile organization. For the sake of anonymity, the company will be referred to as "the company" from this point forward.

A.1 Company Background and Culture

The company has been in existence for roughly ten years starting with a core group of developers and entrepreneurs. The company has been growing quickly over the past two years adding key personnel in managerial, marketing, software development, and quality assurance positions. These new positions have been filled with a mix of seasoned professionals and well educated computer scientists and engineers. The software engineering team consists of the following team roles:

- requirements engineer
- software developers
- quality engineer and testers

- marketing
- customer support and maintenance engineers
- project manager

A newly hired IT security manager coordinates with the team on internal security measures and provides general security consultation but is not part of the software engineering team.

The company culture is entrepreneurial in nature and communication is very open. Physical office space is at a premium and singly occupied offices are rare. Core groups such as developers and testers are all within earshot of each other which aids in open verbal communication. Individuals may have several roles creating cross-functionality among departments or functional areas. There is a shared sense of purpose and direction that creates a sense of *esprit de corps* among all.

A.2 **Product Lifecycle**

The software products developed by the company are cyclical in nature and revolve around a few main "seasons". This leads to very short development life cycles and tight, unbendable deadlines. A typical project timeframe is about 6 months from proposal to delivery. Many of the products are developed as part of a subcontract with mainly one outside developer, but new products developed solely by the company are seen as a future trend.

A.3 Agile Philosophy

As the company grows, the development team sees the need to introduce formality to processes without losing the flexibility that is so much a part of the company culture. In addition, the iterative nature of the product lifecycle has led to a movement to embrace agile philosophies. At this point, a specific agile methodology has not been chosen to follow and general consensus is to use the best practices from several methodologies. The software engineering team has been increasing their agile awareness by attending webinars and through informal research. Currently, they are leaning towards an iterative approach that will fit into the frequent seasonal products that they produce.

A.4 Requirements Process

The requirements process depends on the type of product that is to be developed. There are three primary types of products:

- products developed as part of a contract with another development organization
- products developed as enhancements to existing products
- new products created and developed solely by the company

Contracted Products

Requirements are typically defined prior to contract negotiations. Minor changes may be made during implementation, but requirements are generally not modified.

Existing Products

Typically, customers drive the development of requirements using a request for proposal (RFP). The requirements engineer may make a preliminary requirements document based on the RFP. Final requirements are elicited directly with the customer in consultation with the requirements engineer.

New Products

Requirements are determined by marketing and research in the absence of an existing customer. In this case, stakeholders are all internal, but are from upper management, marketing and high-level developers. Requirements from similar products may be used as a starting point when eliciting and developing requirements, but a requirements repository has not been implemented.

A.5 **Security Needs**

Recently, the company has become more aware of the need to incorporate security into all aspects of the business. Driven initially by systems administration with the backing of top management, there has been an effort to educate everyone regarding security and to develop security policies. This effort seems to have trickled into the psyche of everyone, including the software development team. Developers are aware of the need to incorporate secure coding practices and have been instituting "best practices" into programming. However, these efforts are not driven by any formal processes.

Systems administration has been tasked with developing an overall security roadmap for the company which would include elements such as policy, training,

incident response, and disaster recovery. As the company grows, the need for standardized processed is becoming evident. Standardization should not hinder the entrepreneurial nature and culture of the company and will need to be rolled out in a continuous manner. Certification or adherence to standards may also be desirable.

A.6 **Development and Collaboration Tools**

Open source tools are generally preferred and there are not any formal processes for choosing tools. If a developer, manager or functional unit loosely agrees to use a particular tool, they appear to be able to green light its use. Development is primarily managed using Eclipse and software quality assurance (SQA) is trending towards Bugzilla. One tool that is being utilized company-wide is Egroupware[1], an open source business communication tool. Egroupware consists of modular applications that can be implemented as needed. Key features that are being utilized are general communication components such as calendars and email as well as modules for ticketing, document management and wikis.

Ticketing

A tracker application is used for ticketing for a wide range of functions from general management to specific project management tasks. General management would include systems administration, help desk, and operational tasks. Specific project management tasks include setting up developer responsibilities, testing, and ongoing project communication. For a small company, this tool currently meets their needs.

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

Document Management

There are not any formal document management systems and functional areas have document repository space allocated on company servers and individual computers. Egroupware does provide for document management, but this feature is not currently utilized. Microsoft Word is the default word processing application and document type used for nearly all internal and external documentation.

Wikis

Wikis can be created by any member of the organization. Company policies and software development best practices wikis are under development, but implementation and usage are ad-hoc at the current time.

A.7 **Summary**

The company is small sized, has fast paced development lifecycles and is moving towards agile development. Software engineers are not experts in software security and resources are limited to spend additional time on security training and education. Therefore, security initiatives including secure software development practices must be easy to implement and be developed over time. These characteristics are representative for a typical software development organization and are the basis for our security requirements elicitation approach that will address the needs of a small, agile organization.