Security Requirements Engineering

Nancy Mead

August 2006

ABSTRACT: When security requirements are considered at all during the system life cycle, they tend to be general lists of security features such as password protection, firewalls, virus detection tools, and the like. These are, in fact, not security requirements at all but rather implementation mechanisms that are intended to satisfy unstated requirements, such as authenticated access. As a result, security requirements that are specific to the system and that provide for protection of essential services and assets are often neglected. In addition, the attacker perspective is not considered, with the result that security requirements, when they exist, are likely to be incomplete. We believe that a systematic approach to security requirements engineering will help to avoid the problem of generic lists of features and to take into account the attacker perspective. Several approaches to security requirements engineering are described here and references are provided for additional material that can help you ensure that your products effectively meet security requirements.

THE IMPORTANCE OF REQUIREMENTS ENGINEERING

It comes as no surprise that requirements engineering is critical to the success of any major development project. Some studies have shown that requirements engineering defects cost 10 to 200 times as much to correct once fielded than if they were detected during requirements development [Boehm 88, McConnell 01]. Other studies have shown that reworking requirements, design, and code defects on most software development projects costs 40 to 50 percent of total project effort [Jones 86], and the percentage of defects originating during requirements engineering is estimated at more than 50 percent. The total percentage of project budget due to requirements defects is 25 to 40 percent [Wiegers 03].

A prior study found that the return on investment when security analysis and secure engineering practices are introduced early in the development cycle ranges from 12 to 21 percent, with the highest rate of return occurring when the analysis is performed during application design [Berinato 02]. The National Institute of Standards and Technology (NIST) reports that software that is faulty in security and reliability costs the economy $59.5 billion annually in breakdowns and repairs [NIST 02]. The costs of poor security requirements show that even a small improvement in this area would provide a high value. By the time that an

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

application is fielded and in its operational environment, it is very difficult and expensive to significantly improve its security.

Requirements problems are among the top causes [Charette 05] of why

- projects are significantly over budget
- projects are past schedule
- projects are significantly reduced in scope or are cancelled
- development teams deliver poor-quality applications
- products are not significantly used once delivered

These days we have the further problem that the environment in which we do requirements engineering has changed, resulting in an added element of complexity. Software development occurs in a dynamic environment that changes while projects are still in development, with the result that requirements are in flux from the beginning. This can be due to conflicts between stakeholder groups, rapidly evolving markets, the impact of tradeoff decisions, and so on.

In addition, requirements engineering on individual projects often suffers from the following problems:

- Requirements identification typically does not include all relevant stakeholders and does not use the most modern or efficient techniques.
- Requirements are often statements describing architectural constraints or implementation mechanisms rather than statements describing what the system must do.
- Requirements are often directly specified without any analysis or modeling. When analysis is done, it is usually restricted to functional end -user requirements, ignoring (a) quality requirements such as security, (b) other functional and nonfunctional requirements, and (c) architecture, design, implementation, and testing constraints.
- Requirements specification is typically haphazard, with specified requirements being ambiguous, incomplete (e.g., nonfunctional requirements are often missing), inconsistent, not cohesive, infeasible, obsolete, neither testable nor capable of being validated, and not usable by all of their intended audiences.
- Requirements management is typically weak, with ineffective forms of data capture (e.g., in one or more documents rather than in a database or tool) and missing attributes. It is often limited to tracing, scheduling, and prioritization, without change tracking or other configuration management. Alternatively, it may be limited to the capabilities provided by a specific tool, with little opportunity for improvement.

Quality Requirements

Even when organizations recognize the importance of functional end-user requirements, they often still neglect quality requirements, such as performance, safety, security, reliability, and maintainability. Some quality requirements are nonfunctional requirements, but others describe system functionality, even though it may not contribute directly to end-user requirements.

As you might expect, developers of certain kinds of mission-critical systems and systems in which human life is involved, such as the space shuttle, have long recognized the importance of quality requirements and have accounted for them in software development. In many other systems, however, quality requirements are ignored altogether or treated in an inadequate way. Hence we see the failure of software associated with power systems, telephone systems, unmanned spacecraft, and so on. If quality requirements are not attended to in these types of systems, it is far less likely that they will be focused on in ordinary business systems.

This inattention to quality requirements is exacerbated by the desire to keep costs down and meet aggressive schedules. As a consequence, software development contracts often do not contain specific quality requirements but rather some vague generalities about quality, if anything at all.

SECURITY REQUIREMENTS ENGINEERING

If security requirements are not effectively defined, the resulting system cannot be evaluated for success or failure prior to implementation. (See the Risk Management content area.) When security requirements are considered, they are often developed independently of other requirements engineering activities. As a result, specific security requirements are often neglected, and functional requirements are specified in blissful ignorance of security aspects.

In reviewing requirements documents, we typically find that security requirements, when they exist, are in a section by themselves and have been copied from a generic list of security features. The requirements elicitation and analysis that are needed to get a better set of security requirements seldom take place.

As noted previously, operational environments and business goals often change dynamically, with the result that security requirements development is not a one-time activity. Therefore the activities that we will describe should be planned as iterative activities, as change occurs. Although we describe them as one-time activities for the sake of exposition, you can expect mini life cycles to occur over the course of a project. Much requirements engineering research and practice addresses the capabilities that the system will provide. So a lot of attention is given to the functionality of the system, from the user's perspective, but little attention is given to what the system should not do [Bishop 02]. Users have implicit assumptions for the software applications and systems that they use. They expect them to be secure and are surprised when they are not. These user assumptions need to be translated into security requirements for the software systems when they are under development. Often the implicit assumptions of users are overlooked and features are focused on instead.

Another important perspective is that of the attacker. The attacker is not particularly interested in functional features of the system, unless they provide an avenue for attack. The attacker typically looks for defects and other conditions outside the norm that will allow a successful attack to take place. It's important for requirements engineers to think about the attacker's perspective and not just the functionality of the system from the end-user's perspective. The discussion of attack patterns in Chapter 2 of Software Security Engineering: A Guide for Project Managers [Allen 08] provides a good place to start this analysis. Other techniques that can be used in defining the attacker's perspective are misuse and abuse cases, attack trees [Ellison 03, Schneier 00], and threat modeling. Security requirements are often stated as negative requirements. As a result, general security requirements, such as "The system shall not allow successful attacks," are usually not feasible, as there is no consensus on ways to validate them other than to apply formal methods to the entire system. We can, however, identify the essential services and assets that must be protected. Operational usage scenarios can be extremely helpful aids to understanding which services and assets are essential. By providing threads that trace through the system, operational usage scenarios also help to highlight security requirements, as well as other quality requirements such as safety and performance [Reifer 03]. Once the essential services and assets are understood, we are able to validate that mechanisms such as access control, levels of security, backups, replication, and policy are implemented and enforced. We can also validate that the system properly handles specific threats identified by a threat model and correctly responds to intrusion scenarios.

A discussion of the importance of security requirements engineering can be found at [Mead 08].

Methods and Techniques

As usable approaches to security requirements engineering continue to be developed and mechanisms are identified to promote organizational use, project managers can do a better job of ensuring that the resulting product effectively meets security requirements. Some useful techniques include

- Comprehensive, Lightweight Application Security Process (CLASP) approach to security requirements engineering. CLASP is a life-cycle process that suggests a number of different activities across the development life cycle in order to improve security. Among these is a specific approach for security requirements.
- System Quality Requirements Engineering (SQUARE). This is a process aimed specifically at security requirements engineering.
- Core security requirements artefacts. This approach takes an artifact view and starts with the artifacts that are needed to achieve better security requirements. It provides a framework that includes both traditional requirements engineering approaches to functional requirements and an approach to security requirements engineering that focuses on assets and harm to those assets.

Some other useful techniques are formal specification approaches to security requirements, such as REVEAL and Software Cost Reduction (SCR), and the higher levels of the Common Criteria.

As an additional reference, the SOAR report Software Security Assurance [Goertzel 07] contains a good discussion of SDLC processes and various approaches to security requirements engineering.

In this content area we discuss several approaches, including misuse and abuse cases [process diagram], SQUARE, elicitation and associated case studies, and prioritization and an associated case study. We consider cost/benefit associated with security requirements in two articles. One article considers cost/benefit using a variety of prioritization methods. Another article discusses the use of integer programming for optimizing investment in implementation of security requirements elicitation and security requirements prioritization. While the processes we discuss are similar to those used for requirements engineering in general, we have found that when we get into the detailed steps of how to do security requirements engineering, there are specific techniques that are particularly useful, and we highlight these where they occur. We list local references. A more comprehensive bibliography is also included for this topic.

Although much work remains to be done, organizations can significantly improve the security of their systems by utilizing a systematic approach to security requirements engineering. The methods described here can help in this task.

Maturity of Practices

The techniques described have all had successful pilots and prototypes. SCR, REVEAL, and Common Criteria are mature practices.

BUSINESS CASE RATIONALE

Although data exists to support the benefit of requirements engineering in general, the data to specifically support the benefits of security requirements engineering is anecdotal. The discussion of integer programming for prioritizing investments in security requirements is one such example. Organizations that systematically develop security requirements see benefit from this activity, but it is not usually quantified in terms of return on investment. Organizations that have observed return on investment are typically vendor organizations such as Microsoft. Fortify has done vendor studies showing dramatic return on investment for eliminating vulnerabilities at requirements time, rather than later in the software development life cycle. We hope that in the future more supporting data will be amassed and made available to support this important activity. Discussion of a broader business case development model should be helpful to those striving to develop specific business cases. (See Business Case Models.)

REFERENCES

[Allen 08]	Allen, J. H., Barnum, S., Ellison, R. J., McGraw, G., & Mead, N. R. Software Security Engineering: A Guide for Project Managers. Boston, MA: Addison-Wesley, 2008.
[Berinato 02]	Berinato, Scott. “Finally, a Real Return on Security Spending.” CIO, April 8, 2002.
[Bishop 02]	Bishop, Matt. Computer Security: Art and Science. Boston, MA: Addison-Wesley Professional, 2002.
[Boehm 88]	Boehm, Barry W. & Papaccio, Philip N. “Understanding and Controlling Software Costs. IEEE Transactions on Software Engineering 14, 10 (October 1988): 1462-1477.
[Charette 05]	Charette, R. N. “Why Software Fails.” IEEE Spectrum 42, 9 (September 2005): 42-29.
[Ellison 03]	Ellison, Robert J. & Moore, Andrew. P. Trustworthy Refinement Through Intrusion-Aware Design (CMU/SEI-2003-TR-002, ADA414865). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
[Goertzel 07]	Goertzel, Karen Mercedes; Winograd, Theodore; McKinley, Holly Lynne; Oh, Lyndon; Colon, Michael; McGibbon, Thomas; Fedchak, Elaine; & Vienneau, Robert. Software Security Assurance: A State-of-the-Art Report (SOAR). Herndon, VA: Information Assurance Technology Analysis Center (IATAC) and Defense Technical Information Center (DTIC), 2007.
[Jones 86]	Jones, Capers, ed. Tutorial: Programming Productivity: Issues for the Eighties, 2nd Ed. Los Angeles: IEEE Computer Society Press, 1986.

[Linger 98]	Linger, R. C.; Mead, N. R.; & Lipson, H. F. "Requirements Definition for Survivable Systems," 14-23. Third International Conference on Requirements Engineering. Colorado Springs, CO, April 6-10, 1998. Los Alamitos, CA: IEEE Computer Society, 1998.
[McConnell 01]	McConnell, Steve. “From the Editor - An Ounce of Prevention.” IEEE Software 18, 3 (May 2001): 5-7.
[Mead 08]	Mead, N. R. & Allen, J. H. "Identifying Software Security Requirements Early, Not After the Fact" (audio). InformIT, 2008.
[Mead 03]	Mead, N. R. Requirements Engineering for Survivable Systems (CMU/SEI-2003-TN-013). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
[NIST 02]	National Institute of Standards and Technology. "Software Errors Cost U.S. Economy $59.5 Billion Annually" (NIST 2002-10), 2002.
[Reifer 03]	Reifer, D.; Boehm, B.; & Gangadharan, M. “Estimating the Cost of Security for COTS Software,” 178–186. Proceedings of the Second International Conference on COTS-Based Software Systems. Ottawa, Ontario, Canada, February 2003. Springer, Lecture Notes in Computer Science, 2003.
[Schneier 00]	Schneier, Bruce. Secrets and Lies: Digital Security in a Networked World. New York, NY: John Wiley & Sons, 2000.
[Wiegers 03]	Wiegers, Karl E. Software Requirements. Redmond, WA: Microsoft Press, 2003.