Requirements and Analysis for Secure Software

Software Assurance Pocket Guide Series:
Development, Volume IV
Version 1.0, October 5, 2009

Software Assurance (SwA) Pocket Guide Resources

This is a resource for ‘getting started’ in selecting and adopting relevant practices for delivering secure software. As part of the Software Assurance (SwA) Pocket Guide series, this resource is offered for informative use only; it is not intended as directive or presented as being comprehensive since it references and summarizes material in the source documents that provide detailed information. When referencing any part of this document, please provide proper attribution and reference the source documents, when applicable.

This volume of the SwA Pocket Guide series focuses on requirement and analysis for secure software. It describes the steps and knowledge required to establish the requirements and specifications for secure software, and when to apply them during the Software Development Life Cycle (SDLC).

At the back of this pocket guide are references, limitation statements, and a listing of topics addressed in the SwA Pocket Guide series. All SwA Pocket Guides and SwA-related documents are freely available for download via the SwA Community Resources and Information Clearinghouse at https://buildsecurityin.us-cert.gov/swa.

Acknowledgements

SwA Pocket Guides are developed and socialized by the SwA community as a collaborative effort to obtain a common look and feel and are not produced by individual entities. SwA Forum and Working Groups function as a stakeholder meta-community that welcomes additional participation in advancing software security and refining. All SwA-related information resources that are produced are offered free for public use. Inputs to documents for the online resources are invited. Please contact Software.Assurance@dhs.gov for comments and inquiries. For the most up to date pocket guides, check the website at https://buildsecurityin.us-cert.gov/swa/.

The SwA Forum and Working Groups are composed of government, industry, and academic members and focuses on incorporating SwA considerations in the acquisition and development processes relative to potential risk exposures that could be introduced by software and the supply chain.

Participants in the SwA Forum’s Processes & Practices Working Group collaborated with the Technology, Tools and Product Evaluation Working Group in developing the material used in this pocket guide as a step in raising awareness on how to incorporate SwA throughout the Software Development Life Cycle (SDLC).

Information contained in this pocket guide is primarily derived from the documents listed in the *Resource* boxes that follow throughout this pocket guide.

Special thanks to the Department of Homeland Security (DHS), National Cyber Security Division's Software Assurance team and Nancy Mead, who provided much of the support to enable the successful completion of this guide and related SwA documents.

Resources

» "Requirements Engineering", Nancy R. Mead, DHS Build Security In (BSI) portal at https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/requirements.html.

» "Software Security Assurance: A State-of-the-Art Report"*,* Goertzel, Karen Mercedes, *et al*, Information Assurance Technology Analysis Center (IATAC) of the DTIC at http://iac.dtic.mil/iatac/download/security.pdf .

» Nancy R. Mead, *et al*, "Software Security Engineering: A Guide for Project Managers." Upper Saddle River, New Jersey: Addison-Wesley, 2008.

» "Microsoft Security Development Lifecycle (SDL) – Process Guidance" at http://msdn.microsoft.com/en-us/library/84aed186-1d75-4366-8e61-8d258746bopq.aspx.

Overview

According to data presented by Fortify, the cost of correcting security flaws at the requirements level is up to 100 times less than the cost of correcting security flaws in fielded software. Another study found that the return on investment (ROI) 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. Regardless of the data or study presented, the costs of correcting secure flaws are more expensive to repair or patch the later in the Software Development Life Cycles (SDLC) than those flaws are identified earlier in the SDLC.

Comprehensive requirements are critical for successful system development, but all too often, requirements fail to explicitly consider security. As a result, systems meet the functionality but are rarely safe and consequently are the victim of attacks. Systems which carefully document security requirements reduce the likelihood of successful attacks. Security requirements include functions that implement a security policy such as areas of access control, identification, authentication and authorization, and other functions that perform encryption, decryption, and key management. These functions prevent the violation of the security properties of the system or the information it processes, such as unauthorized access, modification, denial of service, disclosure, *etc.* Security requirements which are testable, complete, unambiguous and measureable will produce more secure software.

The material in this guide approaches requirements from a security perspective. It is assumed that the reader is already familiar with the process of functional software requirements development. Several approaches to security requirements engineering are described in this pocket guide and references are provided for additional material that can help ensure that an organization's products effectively meet security requirements.

The Need for Requirements

Most vulnerabilities and weaknesses in software systems can be traced to inadequate or incomplete requirements or requirements that fail to specify the functions, constraints, and non-functional properties of the software. Software must be dependable, trustworthy, and resilient. If software cannot satisfy these three needs (reliability, trustworthiness, and resiliency) it is of dubious value. Thus, these three needs should be addressed by all of the requirements for the software's functionality, behaviors, and constraints.

Requirements engineering is critical to the success of any major software development project. 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. 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, and the percentage of defects originating during requirements engineering is estimated at more than 50 percent.

Inadequate or Incomplete Requirements Side-effects:

- Projects significantly over budget,
- Projects severely overdue,
- Projects cancellation,
- Project significant scope reduction,
- Poor quality end product, and
- Rarely used product.

According to data presented by Fortify, the cost of correcting security flaws at the requirements level is up to 100 times less than the cost of correcting security flaws in fielded software. A prior study found that the return on investment (ROI) 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. The National Institute of Standards and Technology (NIST) reports software that is faulty in security and reliability costs the economy $59.5 billion annually in breakdowns and repairs. David Rice, former NSA cryptographer and author of ***Geekonomics: The Real Cost of Insecure Software***, approximates that the total economic cost of security flaws is around US $180 billion a year, as reported on Forbes.com. 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 application is fielded and in its operational environment, it is very difficult and expensive to significantly improve its overall security.

On-line Resources

- Nancy R Mead, "Security Requirements Engineering", Build Security In (BSI) portal at https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/requirements/243-BSI.html.
- Barmak Meftah, "Business Software Assurance Identifying and Reducing Software Risk in the Enterprise" at https://buildsecurityin.us-cert.gov/swa/downloads/Meftah.pdf.

Requirements Development

Software requirements by and large are often thought of as requirements for functionality, and in some cases, they are requirements for performance constraints (*i.e.,* the system must have five inputs). They tend to be expressed in positive terms. The Guide to the Software Engineering Body of Knowledge (SWEBOK) defines a software requirement as "a property which must be exhibited in order to solve some problem in the real world." Traditional software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problem.

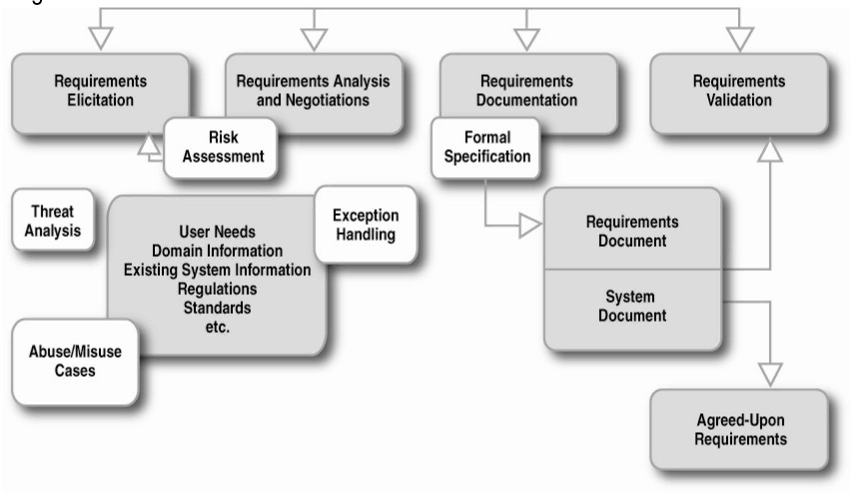
Well-specified Requirements Attributes:

- Testable,
- Complete,
- Unambiguous, and
- Measurable.

There is no single definition for software security requirements. Software security requirements tend to be either constraints on functionality or a statement of a needed property or attribute that will be manifested by the software behavior. Charles Haley *et al,* propose representing security requirements as crosscutting threat descriptions that end up being defined as a set of constraints on the functional requirements. These constraints are the "security requirements." At least initially during requirements capture, security requirements are often stated in negative terms.

There is extensive work on requirements development, as well as tools and techniques to support the processes. Unfortunately, most of the work fails to explicitly consider security. Work that does is usually concerned with security requirements in the sense of requirements development for the security functionality in a system, such as access controls. A large portion of requirements development research and practice addresses the capabilities that the system will provide from the user's perspective, while little attention is given to what the system should not do. 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 while they are under development. Often the implicit assumptions of users are overlooked and the features become the main focus instead. Figure 1 provides an example of additional activities (white background boxes) for increasing software security superimposed over the generic requirements development process.

Figure 1 – Secure Requirements Additions to the Functional Requirement Process



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 developers to think about the attacker's perspective and not just the functionality of the system from the end-user's perspective. A discussion of attack patterns can be found in Chapter 2 of the *Software Security Engineering: A Guide for Project Managers*. Detailed articles and cataloged attack patterns can be found at the BSI and Common Attack Pattern Enumeration and Classification (CAPEC) portals listed on the following resource box.

Other techniques that can be used in defining the attacker's perspective are misuse and abuse cases, attack trees 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. However, essential services and assets that must be protected can be identified. 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. Once the essential services and assets are understood, an organization is able to validate that mechanisms such as access control, levels of security, backups, replication, and policy are implemented and enforced. One can also validate that the system properly handles specific threats identified by a threat model and correctly responds to intrusion scenarios.

On-line Resources

- IEEE Computer Society,"Guide to the Software Engineering Body of Knowledge (SWEBOK)" at http://www2.computer.org/portal/web/swebok.
- Attack Patterns, Build Security In (BSI) portal at https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/attack.html.
- Common Attack Pattern Enumeration and Classification (CAPEC) at http://capec.mitre.org/.
- Charles B. Haley, *et al*, "Deriving Security Requirements From Crosscutting Threat Descriptions", at http://mcs.open.ac.uk/cbh46/papers/AOSD04.pdf.

Requirements Elicitation

Requirements elicitation is the process that addresses where requirements come from and how to collect them (SWEBOK). Using an elicitation method can help in producing a consistent and complete set of security requirements. However, ordinary functional (end-user) elicitation methods such as scenarios, interviews, etc. may not result in a consistent and complete set of security requirements. When security requirements are elicited in a systematic way, the resulting system is likely to have fewer security exposures.

This section provides an overview of a number of elicitation methods and tradeoff analysis to select a suitable one. Companion case studies by Nancy R. Mead can be found at the BSI portal. While results may vary from one organization to another, the discussion of the selection process and various methods should be of general use. Requirements elicitation is an active research area, and is expected to see advances in the future. The expectation within the community is that eventually there will be studies measuring which methods are most effective for eliciting security requirements. At present, however, there is little if any data comparing the effectiveness of different methods for eliciting security requirements.

The following list identifies several methods that could be considered for eliciting security requirements. A description on misuse cases and threat modeling is provided next. For details on the other methods please visit the BSI portal.

- Misuse/Abuse Cases
- Threat Analysis
- Soft Systems Methodology
- Quality Function Deployment
- Controlled Requirements Expression
- Issue-based Information Systems
- Joint Application Development
- Feature-oriented Domain Analysis
- Critical Discourse Analysis
- Accelerated Requirements Method

Misuse/Abuse Cases – Misuse cases are similar to use cases, except that they are meant to detail common attempted abuses of the system. Like use cases, misuse cases require understanding the services that are present in the system. A use case generally describes behavior that the system owner wants the system to show. Misuse cases apply the concept of a negative scenario – that is, a situation that the system's owner does not want to occur. Misuse cases are also known as abuse cases. For an in-depth view of misuse cases, see Gary McGraw's "Misuse and Abuse Cases: Getting Past the Positive" at the BSI portal.

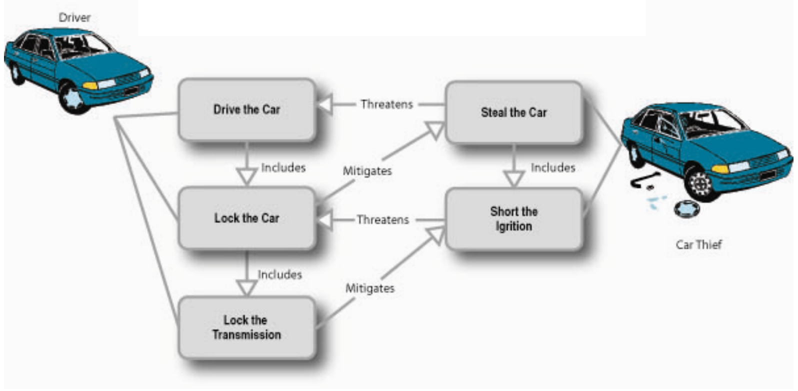


Figure 2 – Misuse Case Example

Misuse cases can help organizations begin to see their software in the same light that attackers do. As use-case models have proven quite helpful for the functional specification of requirements, a combination misuse cases and use cases could improve the efficiency of eliciting all requirements in a system engineering life cycle. Guttorm Sindre and Andreas Opdahl extended use-case diagrams with misuse cases to represent the actions that systems should prevent in tandem with those that they should support for security and privacy requirement. There are several templates for misuse and abuse cases provided by Sindre and Opdahl, and Alexander. Figure 2 is an example of a use/misuse case diagram from Alexander's paper. The high-level case is shown above. Alexander indicated that Misuse and Use Cases may be developed in stages; going from system to subsystem levels and lower as necessary. Lower-level cases may highlight aspects not considered at higher levels, possibly forcing re-analysis. The approach is not rigidly top-down but offers rich possibilities for exploring, understanding, and validating the requirements in any direction.

Use cases describe system behavior in terms of functional (end-user) requirements. Misuse cases and use cases may be developed from system to subsystem levels – and lower as necessary. Lower level cases may draw attention to underlying problems not considered at higher levels and may compel system engineers to reanalyze the system design. Misuse cases are not a top-down method, but they provide opportunities to investigate and validate the security requirements necessary to accomplish the system's mission.

As with normal use cases, one should expect misuse cases to require adjustment over time. Particularly, it is common to start with high-level misuse cases, and refine them as the details of the system are better understood. Determining misuse cases generally constitutes an informed brainstorming activity among a team of security and reliability experts. In practice, the team of experts asks questions of a system's designers to help identify the places where the system is likely to have weaknesses by assuming the role of an attacker and thinking like an attacker would. Such brainstorming involves a careful look at all user interfaces (including environmental factors) and considers events that developers assume a person can't or won't do. There are three good starting points for structured brainstorming:

- » First, one can start with a pre-existing knowledge base of common security problems and determine whether an attacker may have cause to think such vulnerability is possible in the system. Then, one should attempt to describe how the attacker will leverage the problem if it exists.
- » Second, one can brainstorm on the basis of a list of system resources. For each resource, attempt to construct misuse cases in connection with each of the basic security services: authentication, confidentiality, access control, integrity, and availability.
- » Third, one can brainstorm on the basis of a set of existing use cases. This is a far less structured way to identify risks in a system, yet is good for identifying representative risks and for ensuring the first two approaches did not overlook any obvious threats. Misuse cases derived in this fashion are often written in terms of a valid use and then annotated to have malicious steps.

The OWASP CLASP process recommends describing misuse cases as follows:

- » A system will have a number of predefined roles, and a set of attackers that might reasonably target instances of the system under development. These together should constitute the set of actors that should be considered in misuse cases.
- » As with traditional use cases, establish which actors interact with a use case – and how they do so – by showing a communicates-association. Also as traditionally done, one can divide use cases or actors into packages if they become too unwieldy.
- » Important misuse cases should be represented visually, in typical use case format, with steps in a misuse set off (*e.g*., a shaded background), particularly when the misuse is effectively an annotation of a legitimate use case.
- » Those misuse cases that are not depicted visually but are still important to communicate to the user should be documented, as should any issues not handled by the use case model.

Next, identify defense mechanisms for various threats specified in a use case model and update the use case model to illustrate the defense mechanism. If there is no identified mechanism at a particular point in time, the use case should be annotated to say so. Defense mechanisms either should map directly to a functional requirement, or, if the defense mechanism is user-dependent, to an item in an operational security guide.

Finally, evaluate the results with stakeholders and discuss the misuse case with stakeholders, so that they have a clear understanding of the misuse case and agree that it is an adequate reflection of their requirements.

Resources

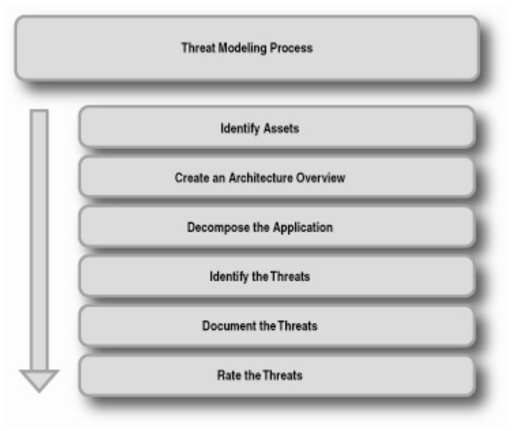
- » Paco Hope, *et al*, "Misuse and Abuse Cases: Getting Past the Positive", BSI portal at https://buildsecurityin.us-cert.gov/daisy/bsi/125-BSI/version/6/part/4/data/bsi2-misuse.pdf?branch=main&language=default.
- » Ian Carter, "Misuse Cases Help to Elicit Non-Functional Requirements" at http://easyweb.easynet.co.uk/~iany/consultancy/misuse_cases/misuse_cases.htm.
- » "Threat Modeling: Diving into the Deep End", Jeffrey A. Ingalsbe, *et al*, at https://buildsecurityin.us-cert.gov/daisy/bsi/resources/articles/932-BSI.html, *IEEE Software*, January/February 2008.

Threat Modeling – A **threat** is a potential occurrence, malicious or otherwise, that might damage or compromise your system resources. Threat modeling is a systematic process that is used to identify threats and vulnerabilities in software. Threat modeling has become a popular technique to help system designers think about the security threats that their system might face. Therefore, threat modeling can be seen as risk assessment for software development. It enables the designer to develop mitigation strategies for potential vulnerabilities and helps them focus their limited resources and attention on the parts of the system most at risk. It is recommended that all software systems have a threat model developed and documented. Threat models should be created as early as possible in the SDLC, and should be revisited as the system evolves and development progresses. To develop a threat model, implement a simple approach that follows the NIST 800-30 [11] standard for risk assessment. This approach involves:

- » **Decomposing the application** – understand, through a process of manual inspection, how the application works, its assets, functionality, and connectivity.
- » **Defining and classifying the assets** – classify the assets into tangible and intangible assets and rank them according to business importance.
- » **Exploring potential vulnerabilities** – whether technical, operational, or management.
- » **Exploring potential threats** – develop a realistic view of potential attack vectors from an attacker's perspective, by using threat scenarios or attack trees.
- » **Creating mitigation strategies** – develop mitigating controls for each of the threats deemed to be realistic. The output from a threat model itself can vary but is typically a collection of lists and diagrams. The OWASP Code

Review Guide at http://www.owasp.org/index.php/Application_Threat_Modeling outlines a methodology that can be used as a reference for the testing for potential security flaws.

The following is a brief description of several threat modeling methodologies. Some provide a tool for automating the process:



Microsoft's Threat Modeling Process – the Microsoft threat modeling allows for the systematic identification and rating of the threats that are most likely to affect the system under development. By identifying and rating threats based on an understanding of the architecture and implementation of the software, an organization can mitigate the threats with appropriate countermeasures in a logical order, starting with the threats that present the greatest risk. The process contains the following steps:

» Identify assets that the systems must protect.
» Create an architecture overview that includes subsystems, trust boundaries, and data flow.
» Decompose the application by creating a security profile to uncover vulnerabilities in the design, implementation, or deployment.
» Identify the threats and potential vulnerabilities from an attacker's perspective.
» Document each threat using a common threat template that defines a core set of attributes to capture for each threat.
» Rate the threats to prioritize and address the most significant threats first.

The output from the Microsoft threat modeling process is a document that allows project team members to understand the threats that need to be addressed and how to address them. The model consists of architecture diagrams, definitions, identified threats and their attributes. The threat rating step in the process uses the Damage potential, Reproducibility, Exploitability, Affected users, and Discoverability (**DREAD**) model to help calculate risk and determine the impact of a security threat. By using the **DREAD** model, an organization can arrive at the risk rating for a given threat by asking the following questions:

» **D**amage potential: How great is the damage if the vulnerability is exploited?
» **R**eproducibility: How easy is it to reproduce the attack?
» **E**xploitability: How easy is it to launch an attack?
» **A**ffected users: As a rough percentage, how many users are affected?
» **D**iscoverability: How easy is it to find the vulnerability?

Microsoft provides a free threat modeling tool available for downloading at http://msdn.microsoft.com/en-us/security/dd206731.aspx, that isn't specifically designed for security experts. It can be used by developers with limited threat modeling experience by providing guidance on creating and analyzing threat models.

Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) Framework – OCTAVE is a framework for identifying and managing information security risks developed by the Software Engineering Institute. It defines an evaluation method that allows an organization to identify important information assets, the threats to those assets, and the vulnerabilities that may expose those assets to the threats. The compilation of information assets, threats, and vulnerabilities helps an organization understand what information is at risk. With this understanding, the organization can design and implement a mitigation strategy to reduce the overall risk exposure of its information assets. OCTAVE can be useful for the following:

- Implementing an organizational culture of risk management and controls.
- Documenting and measuring business risk.
- Documenting and measuring the overall IT security risk.
- Documenting risks surrounding complete systems.
- Implementing a working risk methodology and robust risk management framework.

One of the drawbacks of OCTAVE is that is very large and complex consisting of many worksheets and practices to implement.

Trike – Trike is a framework for security auditing from a risk management perspective through the generation of threat models in a repeatable manner. A security auditing team can use it to describe the security characteristics of a system from its high-level architecture to its low-level implementation details. Trike also enables communication among security team members and between security teams and other stakeholders. The goal of Trike is to automate the repetitive parts of threat modeling. Trike automatically generates threats (and some attacks) based on a description of the system, but this requires that the user describes the system to Trike and check out whether these threats and attacks apply. Trike has similarities to the Microsoft threat modeling processes. However, Trike differs because it uses a risk based approach with distinct implementation, threat, and risk models, instead of using the DREAD aggregated threat model (attacks, threats, and weaknesses).

The Trike tool is available for download at Source Forge website at http://sourceforge.net/projects/trike/files/trike/.

On-line Resources

- Threat Modeling, Microsoft Corp, at http://msdn.microsoft.com/en-us/library/aa302419.aspx.
- Threat Risk Modeling, OWASP, at http://www.owasp.org/index.php/Threat_Risk_Modeling.
- Trike Tools, at http://www.octotrike.org/.

Processes

There are several useful approaches to security requirement development, including the ones examined in this section, and new techniques continue to evolve. These approaches are effective in assisting development teams ensure that the resulting product meets the security requirements. The following is not an exhaustive list of such approaches. Additional ones can be found at the BSI portal.

The Comprehensive, Lightweight Application Security Process (CLASP), sponsored by the Open Web Application Security Project (OWASP), is designed to help software development teams build security into the early stages of existing and new-start software development life cycles in a structured, repeatable, and measurable way. The CLASP is an activity-driven, role-based set of process components guided by formalized best practices. The "Capture Security Requirements" best practice provides a specific approach for security requirements. CLASP strongly recommends that practitioners ensure that security requirements have the same level of "citizenship" as all other "must haves." It's easy for application architects and project managers to focus on functionality when defining requirements, since they support the greater purpose of the application to deliver value to the organization. Security considerations can easily go by the wayside. So it is crucial that security requirements be an explicit part of any application development effort. Among the factors to be considered:

CLASP Steps:

- Detail misuse cases,
- Document security-relevant requirements,
- Identify attack surface,
- Identify global security policy,
- Identify resources and test boundaries,
- Identify user roles and resource capabilities, and
- Specify operational environment.

- An understanding of how applications will be used, and how they might be misused or attacked.
- The assets (data and services) that the application will access or provide, and what level of protection is appropriate given your organization's threshold for risk, regulations you are subject to, and the potential impact on your reputation should an application be exploited.
- The architecture of the application and probable attack vectors.
- Potential compensating controls, and their cost and effectiveness.

Security Quality Requirements Engineering (SQUARE) is another process that provides a means for eliciting, categorizing, and prioritizing security requirements for information technology systems and applications. The focus of this methodology is to build security concepts into the early stages of the development life cycle. The model can also be used for documenting and analyzing the security aspects of fielded systems and for steering future improvements and modifications to those systems. The baseline process is shown in Table 1. In principle, Steps 1-4 are actually activities that precede security requirements engineering but are necessary to ensure that it is successful.

Table 1 – The SQUARE Process

No.	Step	Input	Techniques	Participants	Output
1	Agree on definitions	Candidate definitions from IEEE and other standards	Structured interviews, focus group	Stakeholders, requirements engineer	Agreed-to definitions
2	Identify assets and security goals	Definitions, candidate goals, business drivers, policies and procedures, examples	Facilitated work session, surveys, interviews	Stakeholders, requirements engineer	Assets and goals
3	Develop artifacts to support security requirements definition	Potential artifacts (e.g., scenarios, misuse cases, templates, forms)	Work session	Requirements engineer	Needed artifacts: scenarios, misuse cases, models, templates, forms
4	Perform risk assessment	Misuse cases, scenarios, security goals	Risk assessment method, analysis of anticipated risk against organizational risk tolerance, including threat analysis	Requirements engineer, risk expert, stakeholders	Risk assessment results
5	Select elicitation techniques	Goals, definitions, candidate techniques, expertise of stakeholders, organizational style, culture, level of security needed, cost/benefit analysis, etc.	Work session	Requirements engineer	Selected elicitation techniques
6	Elicit security requirements	Artifacts, risk assessment results, selected techniques	Joint Application Development (JAD), interviews, surveys, model-based analysis, checklists, lists of reusable requirements types, document reviews	Stakeholders facilitated by requirements engineer	Initial cut at security requirements
7	Categorize requirements as to level (system, software, etc.) and whether they are requirements or other kinds of constraints	Initial requirements, architecture	Work session using a standard set of categories	Requirements engineer, other specialists as needed	Categorized requirements
8	Prioritize requirements	Categorized requirements and risk assessment results	Prioritization methods such as Analytical Hierarchy Process (AHP), Triage, Win-Win	Stakeholders facilitated by requirements engineer	Prioritized requirements

Table 1 – The SQUARE Process

No.	Step	Input	Techniques	Participants	Output
9	Inspect requirements	Prioritized requirements, candidate formal inspection technique	Inspection method such as Fagan, peer reviews	Inspection team	Initial selected requirements, documentation of decision-making process and rationale

The **Core Security Requirements Artifacts** is a framework of core security requirements artifacts, which unifies the concepts of the two disciplines of functional requirements development and security requirements development. From functional requirements development it takes the concept of functional goals, which are operationalized into functional requirements, with appropriate constraints. From security requirements it takes the concept of assets, together with threats of harm to those assets. Security requirements must satisfy three criteria: *definition, assumptions and satisfaction*.

- » **Definition of Security Requirement** – Stakeholders want to protect assets from harm and express them as security goals.
- » **Incorporate Assumptions about Behavior** – Analysts must choose a subset of the domain that is relevant. Additionally, the analyst must make some explicit and implicit assumptions on trust, essentially which aspects of the system, like the compiler, are assumed to be trustworthy.
- » **Satisfaction of Security Requirement** – Proofs and other high quality arguments establish the adequacy of the confidence that the security requirement is satisfied.

Security goals aim to protect assets from those threats, and are operationalized into security requirements, which take the form of constraints on the functional requirements.

The framework activities are divided into four stages: 1) Identify functional requirements; 2) Identify security goals; 3) Identify security requirements; and 4) Construct satisfaction arguments.

- » **Stage 1 – Identify functional requirements.** The framework's only requirement is to develop the representation of the system context to be produced, i.e. functional requirements that the system must meet for the stakeholders.
- » **Stage 2 – Identify security goals.** Three steps are required to identify the security goals: 1) Identify candidate assets, 2) select management principles, and 3) Determine security goals. An example of a security goal is "achieve separation of duties when paying invoices."
- » **Stage 3 – Identify security requirements.** Security requirements are constraints on functional requirements that are needed to satisfy a security goal. To determine constraints, identify which security goals apply to which functional requirements and the associated assets that fulfill a particular functional requirement. Multiple iterations of this step might be necessary to generate all the security requirements.
- » **Stage 4 – Construct satisfaction arguments.** During this stage, verification of the security requirements is satisfied by the system as described by context. The two part satisfaction argument with a formal outer argument that is first constructed based on the behavior specifications of the system and informal structured inner arguments constructed to support the assumptions about system composition and behavior.

By first requiring the construction of the formal argument based on domain properties, analysts discover which domain properties are critical for security. Constructing the informal arguments showing that these domain properties can be trusted helps point the analyst towards vulnerabilities, which can be removed through either modification of the problem, addition of security functions or addition of trust assumptions that discount the vulnerability.

On-line Resources

- » CLASP Best Practice 3: Capture Security Requirements at http://www.owasp.org/index.php/Category:BP3_Capture_security_requirements.
- » Nancy R. Mead, "SQUARE Process", BSI portal at https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/requirements/232-BSI.html.
- » Charles H. Haley *et al*, Security Requirements Engineering: A Framework for Representation and Analysis, IEEE Computer Society Volume 34, No 1 January/Febraury2008 at http://www2.computer.org/portal/web/csdl/doi/10.1109/TSE.2007.70754.

Requirements Prioritization

After the security requirements have been indentified, they need to be prioritized. The ultimate goal of every development team is to meet or exceed the stakeholder's needs. However, time and cost constraints put a limit on the number of stakeholder's security requirements that developers can implement. Project decision-makers face the task of selecting the most worthy, practical security requirements that have been elicited and still meet the stakeholder's needs. Prioritization is also effective when security requirements are implemented in stages by providing the opportunity to select which ones to implement first in the order of importance. Most organizations use an informal process that results in software that contains security vulnerabilities. Some organizations select the easiest to implement or lowest cost security requirements without considering their importance. Without a systematic approach that employs effective techniques to make these crucial choices, this outcome is hardly surprising. Despite recent research in security requirements prioritization more work is needed before considering this a mature area. There is a clear need for simple, effective, and industrially proven techniques for prioritizing security requirements.

Clear, unambiguous knowledge about security requirement priorities helps to focus the development process and more effectively and efficiently manage projects. It can also assist in making acceptable tradeoffs among sometimes conflicting goals such as functionality, quality, cost, and time-to-market and to allocate resources based on the security requirement's importance to the project as a whole.

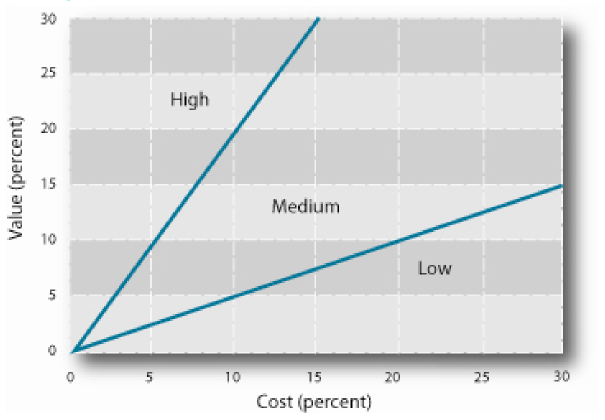
A number of prioritization methods have been found to be useful in traditional requirements development and could potentially be used for security requirements including:

- » Binary Search Tree
- » Numeral Assignment Technique
- » Planning Game
- » The 100-Point Method
- » Theory-W
- » Requirements Triage
- » Wiegers' Method
- » Analytic Hierarchy Process (AHP)
- » Requirements Prioritization Framework

For additional information on these methods please visit the BSI portal a brief description of AHP and Requirements Prioritization Framework follows:

AHP – Karlsson and Ryan developed the AHP process and analytical tool for prioritizing requirements based on a cost-value approach. The AHP tool can help to rank candidate requirements in two dimensions: according to their value to customer and users, and according to their estimated cost of implementation. It uses a pairwise comparison matrix to calculate the value and costs of individual security requirements relative to each other. The pairwise comparison approach includes redundancy and is thus less sensitive to judgmental errors and likely to increase the likelihood that results are reliable. Using this information as input for their decisions, managers can select the requirements to be implemented.

Figure 3 – AHP Relative Cost-Value Sample Plot



The AHP process interprets quality in relation to a requirement's potential contribution to customer satisfaction with the resulting system. *Cost* is the cost of successfully implementing the candidate requirement. In practice, software developers often calculate costs purely in terms of money. However, the authors of AHP found that prioritizing based on relative rather than absolute assignments is faster, more accurate, and more trustworthy.

The steps to prioritizing requirements using the cost-value approach are:

» Requirements engineers carefully review candidate requirements for completeness and without ambiguity.

» Customers and users (or suitable substitutes) apply AHP's pairwise comparison method to assess the relative value of the candidate requirements.

» Experienced software engineers use AHP's pairwise comparison to estimate the relative cost of implementing each candidate requirement.

» A software engineer uses AHP to calculate each candidate requirement's relative value and implementation cost, and plots these on a cost-value diagram. As Figure 3 shows, value is depicted on the *y* axis and estimated cost on the *x* axis.

» The stakeholders use the cost-value diagram as a conceptual map for analyzing and discussing the candidate requirements.

Requirements Prioritization Framework – The requirements prioritization framework and its associated tool includes both elicitation and prioritization activities. This framework is intended to address:

» Elicitation of stakeholders' business goals for the project.

» Rating the stakeholders using stakeholder profile models.

» Allowing the stakeholders to rate the importance of the requirements and the business goals using a fuzzy graphic rating scale.

» Rating the requirements based on objective measure.

» Finding the dependencies between the requirements and clustering requirements so as to prioritize them more effectively.

» Using risk analysis techniques to detect cliques among the stakeholders, deviations among the stakeholders for the subjective ratings, and the association between the stakeholders' inputs and the final ratings.

Comparing Prioritization Techniques – When comparing the available techniques for selecting the most suitable one. Some example evaluation criteria are:

» **Clear-cut steps:** There is clear definition between stages or steps within the prioritization method.

» **Quantitative measurement:** The prioritization method's numerical output clearly displays the client's priorities for all requirements.

- **High maturity:** The method has had considerable exposure and analysis in the requirements engineering community.
- **Low labor-intensity:** A reasonable number of hours are needed to properly execute the prioritization method.
- **Shallow learning curve:** The requirements engineers and stakeholders can fully comprehend the method within a reasonable length of time.

Resources

- Karlsson, J. & Ryan, K. "A Cost-Value Approach for Prioritizing Requirements." *IEEE Software 14,* 5 (September/October 1997): 67-74.

Documenting Security Requirements

The OWASP CLASP document recommends a resource-centric approach to deriving requirements. This approach results in better coverage of security requirements than an ad-hoc or technology-driven methods. For example, many businesses will quickly derive the business requirement “Use Security Sockets Layer (SSL) for security,” without truly understanding what requirements they are addressing. For example, is SSL providing entity authentication, and if so, what is getting authenticated, and with what level of confidence? Many organizations overlook this, and use SSL in a default mode that provides no concrete authentication.

All requirements (not simply security requirements) should be SMART+ requirements, i.e., they should follow a few basic properties:

- ***Specific***. They should be detailed as necessary so that there are no ambiguities in the requirement. This requires consistent terminology between requirements.
- ***Measurable***. It should be possible to determine whether the requirement has been met, through analysis, testing, or both.
- ***Appropriate***. Requirements should be validated, thereby ensuring that they not only derive from a real need or demand but also that different requirements would not be more appropriate.
- ***Reasonable***. While the mechanism or mechanisms for implementing a requirement need not be solidified, one should conduct some validation to determine whether meeting the requirement is physically possible, and possible given other likely project constraints.
- ***Traceable.*** Requirements should also be isolated to make them easy to track/validate throughout the development life cycle.

SMART requirements were originally defined by Mannion and Keepence. OWASP modified the acronym. The original “A” was “Attainable”, meaning physically possible, whereas “Reasonable” was specific to project constraints. The combination of the two requirements was done to focus on appropriateness thus the relabeling of the refinement as SMART+ requirements.

The original paper on SMART requirements has good elaboration on these principles.
See http://www.win.tue.nl/~wstomv/edu/2ip30/references/smart-requirements.pdf.

On-line Resources

- OWASP CLASP v1.2 at http://www.lulu.com/items/volume_62/1401000/1401307/3/print/OWASP_CLASP_v1.2_for_print_LULU.pdf.

Conclusion

This pocket guide compiles requirements and analysis for secures software techniques and offers guidance on when they should be employed during the SDLC. It examines processes, techniques, and tools for the elicitation, prioritization and documentation of software security requirements.

The Software Assurance Pocket Guide Series is developed in collaboration with the SwA Forum and Working Groups and provides summary material in a more consumable format. The series provides informative material for SwA initiatives that seek to reduce software vulnerabilities, minimize exploitation, and address ways to improve the routine development, acquisition and deployment of trustworthy software products. Together, these activities will enable more secure and reliable software that supports mission requirements across enterprises and the critical infrastructure.

For additional information or contribution to future material and/or enhancements of this pocket guide, please consider joining any of the SwA Working Groups and/or send comments to Software.Assurance@dhs.gov. SwA Forums are open to all participants and free of charge. Please visit https://buildsecurityin.us-cert.gov for further information.

No Warranty

Reprints

Software Assurance (SwA) Pocket Guide Series

SwA is primarily focused on software security and mitigating risks attributable to software; better enabling resilience in operations. SwA Pocket Guides are provided; with some yet to be published. All are offered as informative resources; not comprehensive in coverage. All are intended as resources for 'getting started' with various aspects of software assurance. The planned coverage of topics in the SwA Pocket Guide Series is listed:

SwA in Acquisition & Outsourcing

I. Software Assurance in Acquisition and Contract Language
II. Software Supply Chain Risk Management & Due-Diligence

SwA in Development

I. Integrating Security in the Software Development Life Cycle
II. Key Practices for Mitigating the Most Egregious Exploitable Software Weaknesses
III. Risk-based Software Security Testing
IV. Requirements & Analysis for Secure Software
V. Architecture & Design Considerations for Secure Software
VI. Secure Coding & Software Construction
VII. Security Considerations for Technologies, Methodologies & Languages

SwA Life Cycle Support

I. SwA in Education, Training & Certification
II. Secure Software Distribution, Deployment, & Operations
III. Code Transparency & Software Labels
IV. Assurance Case Management
V. Assurance Process Improvement & Benchmarking
VI. Secure Software Environment & Assurance Ecosystem
VII. Penetration Testing throughout the Development Life Cycle

SwA Measurement & Information Needs

I. Making Software Security Measurable
II. Practical Measurement Framework for SwA & InfoSec
III. SwA Business Case & Return on Investment

SwA Pocket Guides and related documents are freely available for download via the DHS NCSD Software Assurance Community Resources and Information Clearinghouse at https://buildsecurityin.us-cert.gov/swa.