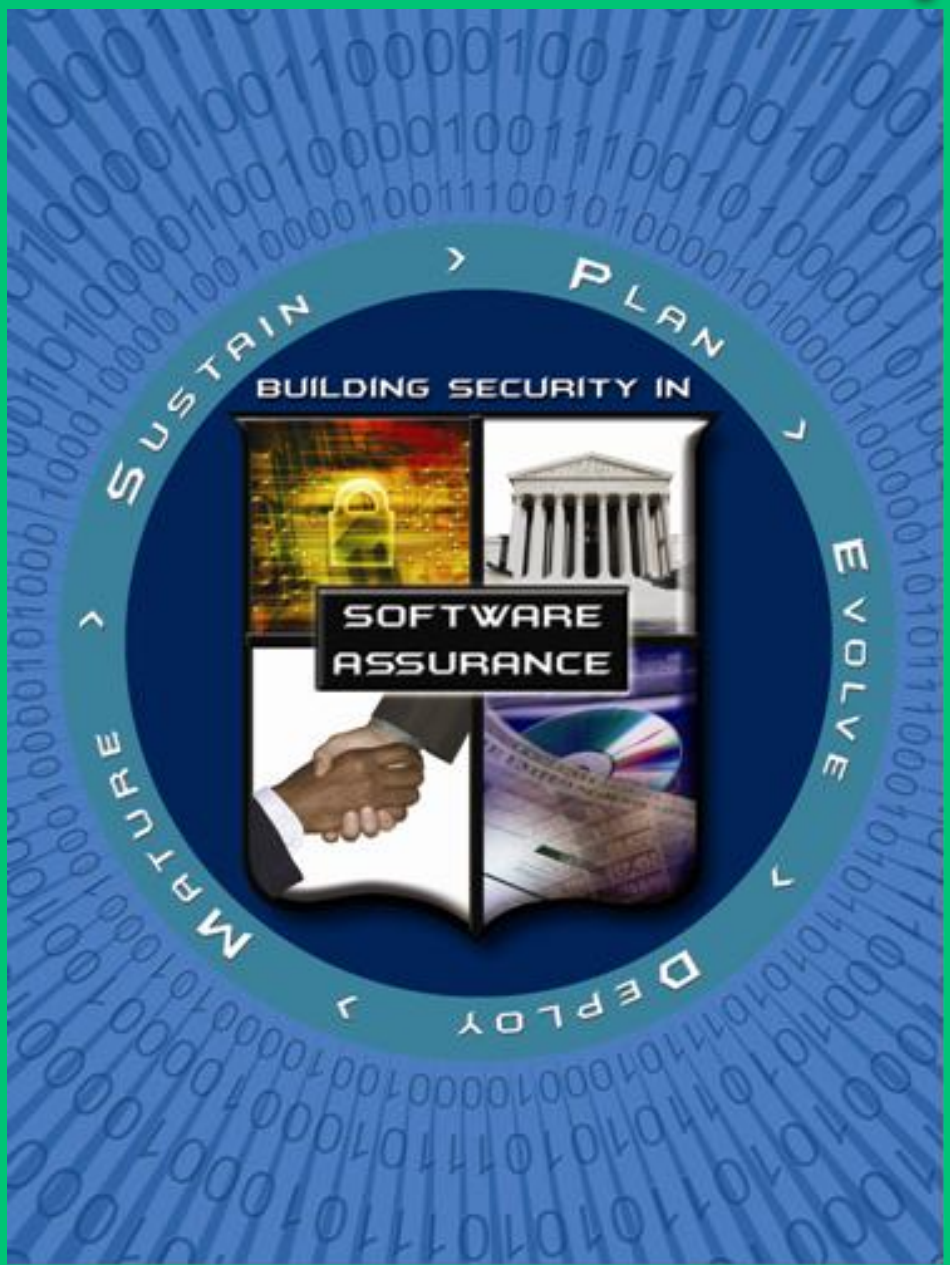
Requirements Analysis for Secure Software

Software Assurance Pocket Guide Series:
Development, Volume IV
Version 2.1, May 18, 2012

4



Software Assurance (SwA) Pocket Guide Resources

This is a resource for 'getting started' in selecting and adopting best 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 comprehensive. Rather it references and summarizes material in the source documents that provide detailed information. When referencing any part of this document, please include attribution to the source documents, when applicable.

This volume of the SwA Pocket Guide series focuses on requirements 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

The SwA community collaborates to develop the SwA Pocket Guides. The SwA Forum and Working Groups function as a stakeholder meta-community that welcomes additional participation in advancing and refining software security. All SwA-related information resources are offered free for public use. The SwA community invites your input: 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/.

Members from government, industry, and academia comprise the SwA Forum and Working Groups. The Working Groups focus on incorporating SwA into acquisition and development processes to manage risk exposure from software and the supply chain.

Participants in the SwA Forum's Processes & Practices Working Group collaborated with the Technology & Tools Working Group in developing the material used in this pocket guide with the goal of raising awareness on how to incorporate SwA throughout the Software Development Life Cycle (SDLC).

Information contained in this pocket guide comes primarily from the documents listed in the *Resource* boxes that appear throughout this pocket guide.

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

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.
- "Software Security Engineering: A Guide for Project Managers." Nancy R. Mead, *et al*,. 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

The evidence is clear: the later flaws are found in the Software Development Life Cycles (SDLC), the more expensive they are to repair or patch. Being no exception, security flaws can be particularly costly.

Comprehensive requirements are critical for successful system development, but, all too often, requirements fail to explicitly consider security. As a result, even fully functional systems, once built, are rarely safe and may be victim to attacks. Conversely, systems that carefully document security requirements reduce the likelihood of succumbing to successful attacks.

Security requirements include functions that implement a security policy in areas of secure coding practices such as: access control, identification, authentication, authorization, 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, or disclosure. Security requirements that are complete, unambiguous, measureable, and testable will produce more secure software.

The material in this guide approaches requirements from a security perspective. Because several security requirements engineering approaches will be covered, it is assumed that the reader is familiar with the process of functional software requirements development. The embedded *Resources* boxes provide additional material on how an organization's products can effectively meet security requirements.

Table of Contents

The Need for Requirements

Software must satisfy the 3 needs:

- Reliability
- Trustworthiness
- Resiliency

Most vulnerabilities and weaknesses in software systems can be traced back to inadequate or incomplete requirements; particularly ones that fail to specify the functions, constraints, and non-functional properties of the software. Typical functionality properties for software are dependability, trustworthiness, and resilience.

Requirements engineering is critical to the success of any major software development project. A study, by Steve McConnell, has shown that requirements engineering defects cost as much as 10 to 200 times to correct, once fielded, than if they were detected during the requirements development. Another study, by Capers Jones, found that reworking requirements, design, and code defects on most software development projects costs 40 to 50 percent of the total project effort, and the percentage of defects originating during requirements engineering is estimated at more than 50 percent.

Once an application is fielded and in its operational environment, it is very difficult and expensive to significantly improve its overall security. According to data presented by Meftah Barmak of 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 as a high value.

Inadequate or Incomplete Requirements Analysis:

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

Resources

- "An Ounce of Prevention", Steve McConnell. http://www.stevemcconnell.com/ieeesoftware/eic17.htm.
- "Estimating Software Costs", Capers Jones. New York: McGraw-Hill, 1998.
- "*Security Requirements Engineering*", Nancy R Mead. Build Security In (BSI) portal at https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/requirements/243-BSI.html.
- "*Business Software Assurance Identifying and Reducing Software Risk in the Enterprise.*" Barmak Meftah. https://buildsecurityin.us-cert.gov/swa/downloads/Meftah.pdf.
- "Geekonomics: The Real Cost of Insecure Software", David Rice. Addison-Wesley Progressional, December 9, 2007.

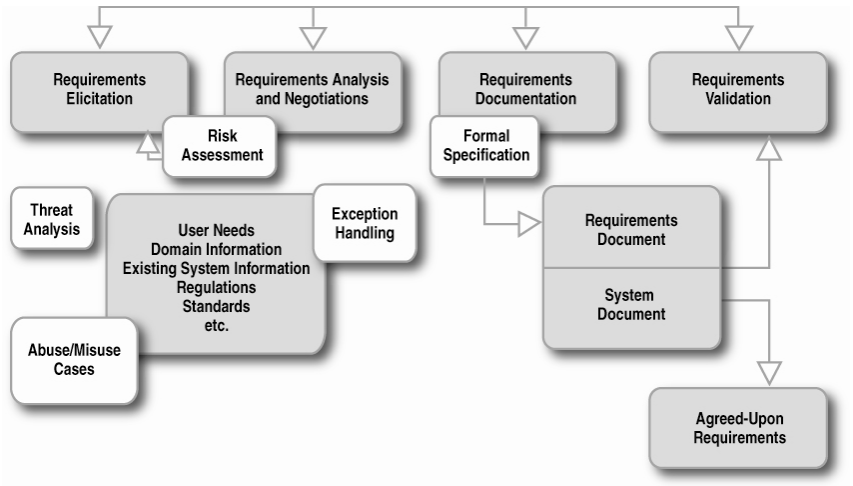
Requirements Development

By and large, software requirements address functionality, and, in some cases, performance constraints (*e.g.* "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 make it contribute to the solution of the given real-world problem.

There is no single definition for software security requirements; they tend to be either constraints on functionality requirements or a statement of a needed property that will be manifested by the software's behavior. Charles Haley *et al,* propose deriving security requirements using aspect-oriented software development crosscutting concepts and problem frames. Security requirements are expressed as constraints on the functional requirements, making them a more natural part of the specification process, comparable with other constraints, such as safety and cost. Security requirements can be expressed as trust assumptions, which indicate that the security requirement is assumed to be satisfied in another context.

There is extensive literature 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 so usually centers on requirements development for the security functionality internal to the 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. Thus, the users expect the software systems 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. Rather, the attacker typically looks for defects and other conditions outside the norm that will allow for a successful attack to take place. It is 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 catalogued attack patterns can be found at the BSI and Common Attack Pattern Enumeration and Classification (CAPEC) portals listed in the Resources box below.

Well-specified requirements attributes are:

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

Other techniques that can be used in defining the attacker's perspective are misuse and abuse cases, attack trees and threat modeling. As noted previously, 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. Instead, essential services and assets that must be protected can and should 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/.
- "Deriving Security Requirements From Crosscutting Threat Descriptions", Charles B. Haley, *et al*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.3590&rep=rep1&type=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 produce a consistent and complete set of security requirements. However, ordinary functional (end-user) elicitation methods, such as scenarios and interviews, may not contribute to the identification of security requirements. When security requirements are identified 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 the tradeoff analysis for selecting a suitable elicitation method. Companion case studies by Nancy R. Mead can be found at the BSI portal, such as "Requirements Elicitation Introduction" which is listed in the following Resources box. 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, propelled by studies measuring which methods are most effective for identifying security requirements. At present, however, there is little if any data comparing the effectiveness of different methods for eliciting security requirements.

Security requirements address:

- Internal and external threats
- Implicit user expectations
- The attacker's mindset

The following list identifies several methods for eliciting security requirements.

- 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

On-line Resources

- Nancy R. Mead, "Requirements Elicitation Introduction", Build Security In (BSI) portal at https://buildsecurityin.us-cert.gov/Build Security In (BSI)/articles/best-practices/requirements/533-BSI.html.

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 an understanding of 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 the following Resource box for Gary McGraw's "Misuse and Abuse Cases: Getting Past the Positive" article at the BSI portal.

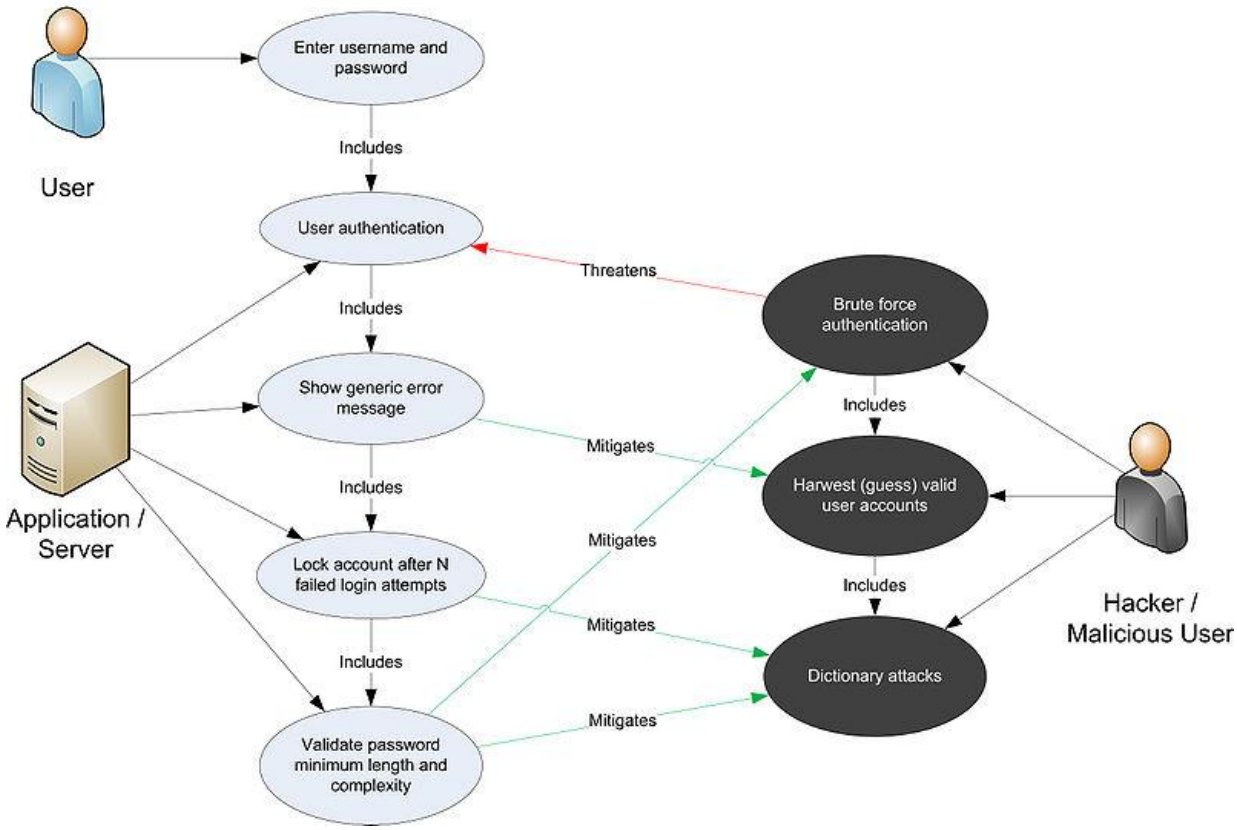
Misuse cases help organizations see their software in the perspective of an attacker. As use-case models have proven quite helpful for the functional specification of requirements, a combination of misuse cases and use cases could improve the efficiency of eliciting all requirements in a system engineering life cycle. Guttorm, Sindre, and Andreas Opdahl provided several templates extending 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. Figure 2 shows an example of a use/misuse case diagram from OWASP's Testing Guide. The use case diagram demonstrates the actions that both the user and the application perform in the particular scenario. The misuse case diagram demonstrates the actions that can be taken by an attacker to take advantage of the system in the particular scenario. The two are linked together by arrows showing which of the attacker's actions threaten the behavior of the user/application as well as which of the user/application's actions thwart the attacker. Making a misuse/abuse cases like this can point out possible security holes in the system.

Use cases describe system behavior in terms of functional (end-user) requirements. Misuse cases provide opportunities to investigate and validate security 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 reassess the system design.

As with normal use cases, misuse cases 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. Misuse cases are typically the result of brainstorming sessions by a team of security and reliability experts and application domain experts. In practice, the team of experts asks questions of system's designers to help identify the places where the system is likely to have weaknesses. These questions help the system's designers think in the way an attacker would. Such brainstorming activity 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 can exploit such vulnerabilities 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, 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 it is good for representing 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.

Figure 2 – Misuse Case Example



The OWASP Comprehensive, Lightweight Application Security Process (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 – this is commonly known as 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 diagrammed in typical use case format, with misuse steps set apart (*e.g.*, using a shaded background), particularly when the misuse is effectively an annotation of a legitimate use case.
- Document separately any remaining issues and any interesting misuse cases that were not diagrammed.

Next, identify and include defense mechanisms for various specific threats into the relevant use case . If there is no identified mechanism at a particular point in time, the use case should 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 with stakeholders the steps to mitigate the misuse case relative to the risk the case presents. Discuss each 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

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

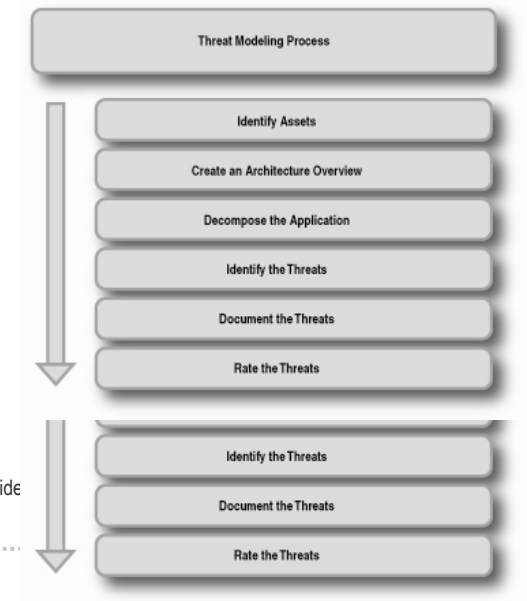
Threat Modeling – A threat is a potential occurrence, malicious or otherwise, that might damage or compromise your system. Threat modeling is a systematic process that is used to identify threats and vulnerabilities in software. Threat modeling has become a popular technique for system designers think about the security threats that their system might encounter, serving as a risk assessment for software development. It enables the designer to develop mitigation strategies for potential vulnerabilities and to focus their limited resources and attention on the parts of the system most at risk. All software systems should develop and document a threat model. Threat models should be created as early as possible in the SDLC, and should be revisited as the system evolves and as development progresses. The National Institute of Standards and Technology (NIST) 800-30 standard for risk assessment can be used as a guideline in developing a threat model. 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 testing for potential security flaws.

The following is a brief description of several threat modeling methodologies, some of which provide tools for automating the process.

Microsoft's Threat Modeling Process – The Microsoft threat modeling process allows for the systematic identification and rating of 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

Figure 3 – Microsoft's Threat Modeling Process



* Refer to the Architecture and Design Considerations for Secure Software pocket guide

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 resulting product from the Microsoft threat modeling process is a document that allows the project team members to understand threats that need to be addressed and ways 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://www.microsoft.com/security/sdl/adopt/threatmodeling.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, developed by the Software Engineering Institute (SEI), defines a set of self-directed activities for organizations to identify and manage their information security risks. 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 it is very thorough and complex, consisting of many worksheets and practices, and hence requires substantial resources 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 Trike to describe the security characteristics of a system from its high-level architecture to its low-level implementation details. Trike also enables communication between security team members and communication between the 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 the user to describe the system to Trike and also check whether these threats and attacks apply. Trike's likeness to Microsoft threat modeling processes diverges due to its use of 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.
- Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) Framework, Version 1.0, at http://www.sei.cmu.edu/library/abstracts/reports/99tr017.cfm.
- Trike Tools, at http://www.octotrike.org/.

Processes

There are several useful process oriented approaches to security requirement development examined in this section. These approaches are effective in assisting development teams ensure the resulting product maximizes security to the extent practicable. The following is not an exhaustive list of such approaches and new techniques continue to evolve. Additional methods to security requirement development can be found at the BSI portal under Requirements Engineering.

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.

The Comprehensive, Lightweight Application Security Process (CLASP) – sponsored by the Open Web Application Security Project (OWASP), CLASP is designed to help software development teams build security into the early stages of existing and newly-started software development life cycles in a structured, repeatable, and measurable way. CLASP is an activity-driven, role-based set of process components guided by formalized best practices. For example, 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 the other "must haves." As stated above, 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. As they are seen as costs, 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:

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

Security Quality Requirements Engineering (SQUARE) – SQUARE is a process that provides a means for eliciting, categorizing, and prioritizing security requirements for information technology systems and applications. This methodology focuses on building 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 these systems. The baseline process is shown in Table 1. In principle, Steps 1-4 are activities that precede security requirements engineering but they are necessary to ensure the process is successful. Versions of SQUARE for privacy and acquisition have also been developed. Reports on SQUARE and tools for download can be found at http://www.cert.org/sse/square/ .

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

Table 1 – The SQUARE Process					
No.	Step	Input	Techniques	Participants	Output
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
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

Core Security Requirements Artifacts (CSRA) –CSRA documents and unifies functional requirements development and security requirements development into one framework. Functional goals can become functional requirements, with appropriate constraints. Security requirements' concepts identify assets and define threats as harm to those assets. According to CSRA, security requirements must satisfy three criteria: *definition, assumptions and satisfaction*.

- **Definition of Security Requirement** – Stakeholders express security goals by describing assets to protect.
- **Assumptions about the Behavior** – Analysts must choose a subset of the domain that is relevant to the system. Additionally, the analyst must make some explicit and implicit assumptions on trust, essentially naming 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 threats, and are operationalized into security requirements, which take the form of constraints on the functional requirements.

The CSRA framework activities are divided into four stages:

- **Stage 1 – Identify functional requirements.** Develop the representation of the system to be produced, e.g. describe what the system must do for the stakeholders.
- **Stage 2 – Identify security goals.** Perform three steps:
  - Step 1) Identify candidate assets, e.g., interactive interface available to employees during working hours
  - Step 2) Select management principles, e.g., all managers submit invoices
  - Step 3) Determine security goals, e.g., achieve separation of duties when paying invoices.1
- **Stage 3 – Identify security requirements.** Security requirements are constraints on functional requirements. The constraints are needed to satisfy a security goal. To determine constraints, identify which security goals apply to which functional requirements and, by extension, to 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 satisfaction argument has two parts:
  - Part 1) Construct the formal outer argument based on the behavior specifications of the system (and the domain properties)
  - Part 2) Construct the informal structured inner arguments to support the assumptions about system composition and behavior.

CSRA analysts must first construct formal arguments based on domain properties in order to discover which domain properties are critical for security. Constructing the informal arguments then shows that these domain properties can be trusted and 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.

The Security Requirements Engineering Process (SREP) – SREP is an asset-based and risk-driven method for establishment of security requirements in the development of secure Information Systems and whose focus seeks to build security concepts at the early phases of the development life cycle. It describes how to integrate security requirements into the software engineering process in a systematic and intuitive way. The SREP approach is based on the integration of the Common Criteria (CC) into the SDLC with the use of a security resources repository to support reuse of security requirements, assets, threats, and counter measures.

More Processes – For additional secure software development processes, please read the "Survey on Requirements and Design Methods for Secure Software Development" report. The report analyze and compares various secure software development processes based on a number characteristics that the processes should have. It also looks at security requirements engineering processes, state-of-the-art secure design languages, secure design guidelines, and will provided guidelines for software developers on how to select specific methods that will fulfill their secure software applications needs.

On-line Resources

- "Requirements Engineering", Build Security In (BSI) portal at https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements.html.
- CLASP Best Practice 3: Capture Security Requirements at http://www.owasp.org/index.php/Category:BP3_Capture_security_requirements.
- "SQUARE Process", Nancy R. Mead. Build Security In (BSI) portal at https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/requirements/232-BSI.html.
- "Security Requirements Engineering: A Framework for Representation and Analysis", Charles H. Haley et al. IEEE Computer Society Volume 34, No 1 January/Febraury2008 at http://www2.computer.org/portal/web/csdl/doi/10.1109/TSE.2007.70754.
- "Core Security Requirements Artefacts." Jonathan D. Moffet, et al. http://computing-reports.open.ac.uk/2004/2004_23.pdf.
- "On Selecting Appropriate Development Processes and Requirements Engineering Methods for Secure Software." Khan, M.U.A., Zulkernine, M. Computer Software and Application Conference, 2009 . http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5254044

Requirements Prioritization

After the security requirements have been identified, they will 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 can limit the number of stakeholder's security requirements the developers can implement. Project decision-makers face the difficult task of selecting the most worthy and practical security requirements that have been elicited and still meets 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 containing security vulnerabilities. Some organizations select the easiest to implement or lowest costing security requirements without considering the impact or probability of the risks they are to address. Without a systematic approach that employs effective techniques to make crucial choices, the outcome of vulnerable software 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 to manage projects more effectively and efficiently. 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 requirements importance to the project as a whole.

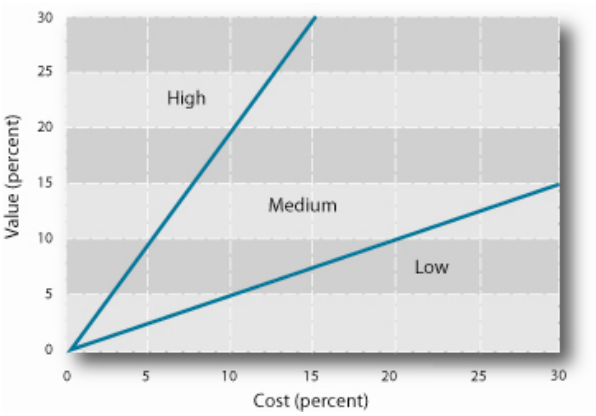
Analytic Hierarchy Process (AHP) – Karlsson and Ryan developed the AHP process and analytical tool for prioritizing requirements based on a cost-value approach. The AHP tool helps rank candidate requirements in two dimensions: according to their value to the customer and users, and according to their 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, making it less sensitive to judgmental errors, thus increasing the likelihood that the results are reliable. Managers can use the pairwise comparison method as input for their decisions to properly select the requirements to be implemented.

The AHP process interprets quality in relation to a requirement's potential contribution to the customer's 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 freedom from 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 4 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.

Figure 5 — AHP Relative Cost-Value Sample Plot



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.

Several 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 for the Nancy R. Mead's "Requirements Prioritization Introduction" article.

Comparing Prioritization Techniques – When comparing several prioritization techniques, use evaluation criteria such as:

- **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 to and feedback from 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

- "Requirements Priorization Introduction", Nancy R. Mead. Build Security In (BSI) portal at https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/545-BSI.html.
- "A Cost-Value Approach for Prioritizing Requirements." Karlsson, J. & Ryan, K. *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 method. 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 as 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 are 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 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 https://www.owasp.org/index.php/Category:OWASP_CLASP_Project#CLASP_v1.2
- "SMART Requirements" Mike Mannion, Barry Keepence. SEI http://www.win.tue.nl/~wstomv/edu/2ip30/references/smart-requirements.pdf

Questions to Ask Developers

The following are questions managers in development and procurement organizations should ask to determine if the requirements engineers are harnessing the techniques for developing requirements for a secure software system. These questions highlight the major areas of security in requirements analysis. Intended to raise awareness of the content of this Guide, they are not a complete set of questions. For a more comprehensive set of questions, reference the "Software Assurance in Acquisition and Contract Language" and "Software Supply Chain Risk Management & Due-Diligence" pocket guides.

- How do the requirements allow for mitigation of known vulnerabilities of the system and what is being done to mitigate the known vulnerability?
- What types of methods are being used to elicit the security requirements?
- What types of tradeoff analysis are being used to select the elicitation method?
- How are the attacker's perspective incorporated into the development of the requirements?
- Are all the requirements feasible and able to be validated?
- What types of processes are being used to develop security requirements?
- How are the security requirements being prioritized?

- What types of prioritization methods are being used?
- Are all the requirements SMART+ requirements?
- Are threads being used to trace through the system operational usage scenarios?

Conclusion

This pocket guide compiles secure software techniques for requirements analysis and offers guidance on when and how 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 the development of more secure and reliable software that supports mission requirements across enterprises and protects 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

This material is furnished on an "as-is" basis for information only. The authors, contributors, and participants of the SwA Forum and Working Groups, their employers, the U.S. Government, other participating organizations, all other entities associated with this information resource, and entities and products mentioned within this pocket guide make no warranties of any kind, either expressed or implied, as to any matter including, but not limited to, warranty of fitness for purpose, completeness or merchantability, exclusivity, or results obtained from use of the material. No warranty of any kind is made with respect to freedom from patent, trademark, or copyright infringement. Reference or use of any trademarks is not intended in any way to infringe on the rights of the trademark holder. No warranty is made that use of the information in this pocket guide will result in software that is secure. Examples are for illustrative purposes and are not intended to be used as is or without undergoing analysis.

Reprints

Any Software Assurance Pocket Guide may be reproduced and/or redistributed in its original configuration, within normal distribution channels (including but not limited to on-demand Internet downloads or in various archived/compressed formats).

Anyone making further distribution of these pocket guides via reprints may indicate on the back cover of the pocket guide that their organization made the reprints of the document, but the pocket guide should not be otherwise altered. These resources have been developed for information purposes and should be available to all with interests in software security.

For more information, including recommendations for modification of SwA pocket guides, please contact Software.Assurance@dhs.gov or visit the Software Assurance Community Resources and Information Clearinghouse: https://buildsecurityin.us-cert.gov/swa to download this document either format (4"x8" or 8.5"x11").

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. Software Security Testing
IV. Requirements Analysis for Secure Software
V. Architecture & Design Considerations for Secure Software
VI. Secure Coding
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

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.