Security-oriented Point-wise Changes to SWEBOK

Samuel T. Redwine, Jr.

James Madison University
Harrisonburg, VA 22807

redwinst@jmu.edu

1 February 2009

Introduction

Background

When the SWEBOK project was originally designed in 1998, the world was not as ubiquitously networked as today (or, at least, most did not perceive it that way). So information on software security was regarded as a specialty topic, hence, not meeting the SWEBOK criterion of "useful most of the time on most projects". Little information on software security was included beyond the admonition to avoid coding practices leading to buffer overruns.

Today, wide agreement exists that a great many software applications require some level of security and that each software project should perform appropriate analysis of the security requirements and perform practices to support the achievement of those requirements. Thus it is prudent for the SWEBOK to include appropriate security information.

During 2009, a SWEBOK refresh will be initiated that will complete in 2010. In order to have a quick start in dealing with security, Sam Redwine of James Madison University has produced this set of point-wise additions to the existing SWEBOK Guide. To allow standalone understanding, this document consists of or short series of sentences enumerating key knowledge (concerns, facts, methods, activities, roles, issues, etc.) at the same level of abstraction as the 2004 edition of SWEBOK. They provide coverage of software security that is listed by SWEBOK sections and that adds to the contents of SWEBOK 2004. In some cases these sentences may later become phrases or list entries when fully integrated into the refreshed SWEBOK's text.

Eventually, this material as well as a supplementary KA on Software Security will be reviewed as part of the SWEBOK consensus process and appropriate material incorporated into the Guide.

However, there is a need to "lean forward" on this topic, not only because of public need for the information, but for the very practical reason that item-writers for the imminent CSDP refresh will also need the material. The point-wise additions listed below will be supplied to the CSDP item-writers.

At the end of this document, the References section lists three textbooks as well as one item for Further Reading.

Security

[[1] Chapters 1 and 2]

Security has become a normal, widespread, and important concern for most software. Like safety, it concerns possible losses and risks. In addition to the traditional issues in safety, which focus on inadvertent and unintentional losses and risks, security must also deal with maliciousness, illegitimacy, and concern for confidentiality possibly including protecting privacy and intellectual property.

Table 1: Some Kinds of Attackers

Amateur hackers
Commercial competitors
Individual criminals, small criminal groups
Insiders
Nation states
Organized crime syndicates
Psychopath or sociopath
Social protestors (hactivists)
Terrorists

While the existence of maliciousness removes no non-malicious problems, it does introduce additional problems. Among these is the non-probabilistic behavior of intelligent, malicious, and often well-resourced attackers. Some of the kinds of attackers of computing systems via exploiting software are listed in Table 1.

Attackers attack because of the value to them from exploiting information, denial, or damage. The value of information to its owner can be quite different from its value to an attacker.

Software is in danger all its life from exploitable requirement and design decisions through the insertion of source code vulnerabilities and attack during deployment and transition to attacks during operation and use and even during retirements and disposal. Unfortunately, the situation is asymmetrical. The attacker has a choice of where and when to attack, and the defender needs to defend against them all. However, resource and other constraints normally mean not everywhere can be defended well. Nevertheless, defenders can try to avoid offering opportunities to attackers (aka vulnerabilities) and make arrangements to tolerate attacks including limiting damage and recovering quickly.

In addition to concern for "real-world" consequences and losses, information security is especially important in software engineering. The acronym CIA is often used to stand for Confidentiality, Integrity, and Availability – three of the important qualities related to information security. Three definitions or characterizations of [information] security appearing in international standards are:

1. ISO/IEC 25010 and ISO/IEC 15026:1998 define security as, "The protection of system items from accidental or malicious access, use, modification, destruction, or disclosure."
2. ISO/IEC 25010 associates the following qualities with security: confidentiality, integrity, non-repudiation, accountability, authenticity, security compliance, and immunity (the degree to which the software product is resistant to attack), plus related survivability and safety.

3. ISO/IEC13335-1 states, “All aspects related to defining, achieving, and maintaining confidentiality, integrity, availability, accountability, authenticity, and reliability.”

Briefly, the confidentiality, integrity, availability, and accountability can be defined as:

- Confidentiality: preservation of authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information.”
- Integrity: lack of improper modification, alteration, or destruction.
- Availability: continually, reliably accessible and usable in a timely manner.
- Accountability: being able to associate actors with their acts; to include non-repudiation (ensuring actors are unable to effectively deny (repudiate) an action or event).

Security of information often requires the simultaneous existence of 1) availability for authorized actions only, 2) confidentiality, and 3) integrity with "improper" meaning "unauthorized".

Organization of Document

The remainder of this document mainly consists of sentences enumerating key knowledge (concerns, facts, methods, activities, roles, issues, etc.) that coverage of software security would add to the contents of SWEBOK 2004. The items are organized by major SWEBOK sections and more specific relevant locations within SWEBOK are given in parentheses. Many of these sentences are stated in the terms already used for similar concerns. While a number of points are relevant to multiple sections, only a few have been duplicated.

Introduction to the Guide

[[2] Chapter 1]

Computer security subsumes software security. Security engineering is a related discipline. While ethics, law, and governance are relevant for other aspects of software engineering, they are particularly relevant to security.

Requirements

[[1] Chapter 3, [2] Sections 1.1, 1.2, 1.6. 1.7, 19.1, 29.2, [3] Section 9.3]

Among the ways requirements can be further classified is as "security requirements." (SWEBOK; Chapter 2, page 2-2, section 1.2, last paragraph)

Security-relevant requirements include requirements for: (SWEBOK; Chapter 2, page 2-2, section 1.3)

- Detection, recording, and reporting (possibly including real-time notification) of malware, attempted attacks, security violations, and possibly scouting and other suspicious behaviors.
- Secure resilience and secure failure.

- Support for security incident response and investigation, and possibly forensics and prosecution.
- Identity management and performing authorizations.
- Protection of intellectual property.
- Protection of the software itself.

During the Requirements Process (SWEBOK; Chapter 2, page 2-3, section 2 and as individually noted), one should:

- Identify stakeholder security-related needs, asset protection needs and sensitivity levels, threats, levels of uncertainty and consequence required, and interface and environment requirements; and usability, reliability, availability, tolerance, survivability, maintainability, deception, evaluatability needs including for certification, accreditation, and auditing as well as trust management concerns. (SWEBOK; Chapter 2, page 2-4, section 3)
- Do analyses including threat analysis; security-related risk analysis and feasibility analysis; analysis of conflicts among security needs; and tradeoff analysis of security properties with others. (SWEBOK; Chapter 2, page 2-6, section 4)
- Document security-related assumptions. (SWEBOK; Chapter 2, page 2-7, section 5)
- Identify software-related security objectives. (SWEBOK; Chapter 2, page 2-7, section 5)
- Specify software-related security policy. (SWEBOK; Chapter 2, page 2-7, section 5)
- Identify needed security functionality. (SWEBOK; Chapter 2, page 2-7, section 5)
- Perform validation of security requirements. (SWEBOK; Chapter 2, page 2-8, section 6)

Attackers are important process actors, but they usually must be treated indirectly, say by consulting security experts and existing documents, and their interests opposed rather than supported. (SWEBOK; Chapter 2, page 2-3, section 2.2, 2nd paragraph)

Stakeholders can state security-related requirements in terms of limitations on losses and their timing. (SWEBOK; Chapter 2, page 2-4, section 2.3, last paragraph)

Security-related requirements can come from requirements to comply with laws, regulations, planned certifications, standards, and organizational policies and procedures including ones related to intellectual property and privacy or personally identifiable information. (SWEBOK; Chapter 2, page 2-5, section 3.1)

A great many operational environments face security threats. (SWEBOK; Chapter 2, page 2-5, section 3.1, 4th bullet)

Organizational environments can affect the number, nature, and success of attacks by insiders. (SWEBOK; Chapter 2, page 2-5, section 3.1, 5th bullet)

A key set of information to elicit/analyze is the uncertainty that is acceptable or tolerable by stakeholders in the achievement of:

- Specifications that will meet their security-related needs and expectations.
- Software that as built, and possibly as transitioned, maintained, operated, etc., meets its security-related specifications such that they will have the confidence they require in the software in the face of the risks or possible consequences they face. (SWEBOK; Chapter 2, page 2-4, section 3, and/or Chapter 2, page 2-6, section 4)

Security-related requirements are a possible subject for requirements negotiations, as many tradeoffs exist with other aspects or qualities of the software and possible limitations on uncertainties can vary e.g. desired versus tolerable. (SWEBOK; Chapter 2, page 2-7, section 4.4)

Precise description of the relevant requirements specifications is particularly important for security-critical software and this has implications for selecting the notation(s) used. (SWEBOK; Chapter 2, page 2-8, section 5.3, 5th paragraph)

A requirements specification needs to call for or be consistent with the requirements that essential security functionality cannot be bypassed and whenever conditions necessitate it, appropriate security functionality must always be invoked. (SWEBOK; Chapter 2, page 2-8, section 5.3, between 5th and 6th paragraph)

The testing and verification methods need to be adequate to provide (along with other evidence) adequately low uncertainties related to the achievement of security-related properties, in order to provide acceptable or tolerable risks and grounds for the needed stakeholder confidence. (SWEBOK; Chapter 2, page 2-8, section 6, 2nd paragraph)

At a fundamental level, security requirements may change slowly, but in the details they tend to change rapidly. (SWEBOK; Chapter 2, page 2-9, section 7)

While the requirements process is iterative, security properties need to be addressed early because of their widespread implications on requirement specifications, partially as a result of their being emergent properties. (SWEBOK; Chapter 2, page 2-10, section 7.1)

Software Design

[[1] Chapters 4 and 6, [2] Sections 4.4, 13, 19.1.2, 19.2, 23.1-2, 29.3, [3] Sections 30.3 and 30.4]

Among the enabling techniques for design are the principles of: (SWEBOK; Chapter 3, page 3-2, section 1.4, new 1.4.7)

- Least privilege – permitting access only to entities with legitimate need and only while it is needed.
- Open design – open to adequate review (i.e., avoiding secrecy by obscurity).

Key issues in software design include security (for coverage of its definition see the Introduction above), i.e. preventing unauthorized disclosure, creation, changing, deleting or denying of information and other resources; and tolerating security-related attacks or violations by limiting damage, continuing service, speeding repair and recovery, and failing and recovering securely. Access control is fundamental to much of security. Also, one must ensure the proper use of cryptology. (SWEBOK; Chapter 3, page 3-3, section 2 and section 2.4)

Design must ensure that security functionality cannot be bypassed and that necessary security functionality is always invoked when appropriate. (SWEBOK; Chapter 3, page 3-3, section 2.2)

Security is a quality attribute of software as well as is the usability of the security functions. (SWEBOK; Chapter 3, page 3-4, section 4.1)

Software design reviews can examine security including performing vulnerability analysis. (SWEBOK; Chapter 3, page 3-4, section 4.2, 1st bullet)

Installer, operator and user aids (e.g. manuals and help files) can be reviewed to ensure that they include security considerations. (SWEBOK; Chapter 3, page 3-4, section 4.2, 1st bullet and/or Chapter 7, page 7-6, section 2.1.6)

Design vulnerability analysis (static analysis for security weaknesses) can be performed if security is a concern. (SWEBOK; Chapter 3, page 3-4, section 4.2, 2nd bullet)

Usually, only certain deployed configurations are secure. (SWEBOK; Chapter 3, page 3-4, section 5.1, 5th bullet and/or Chapter 7)

Data flows (and therefore possibly data-flow diagrams) are important to security as possible paths for attack and disclosure of confidential information. Thus, ensuring appropriate separations is crucial. (SWEBOK; Chapter 3, page 3-5, section 5.2, 3rd bullet)

Behavioral descriptions can include a rationale for why design will meet security requirements. (SWEBOK; Chapter 3, page 3-5, section 5.2)

The amount of software that must be trusted for security needs to be minimized. (SWEBOK; Chapter 3, page 3-5, section 6.1)

Reused and off-the-shelf software should meet the same security requirements as new software. (SWEBOK; Chapter 3, page 3-6, section 6.5 and Chapter 4, section 3.4)

Trust management is a design concern; components treated as having a certain degree of trustworthiness cannot depend on less trustworthy components or services. (SWEBOK; Chapter 3, page 3-6, section 6.5)

Formal methods are highly useful for high security systems. (SWEBOK; Chapter 3, page 3-6, section 6.6)

Software Construction

[[1] Chapter 5, [2] Sections 16.3, 19.3, 29.5]

The choices of allowable programming language subsets and usage standards are important aids in achieving higher security. (SWEBOK; Chapter 4, page 4-2, section 1.4)

Construction languages and their implementations (e.g. compilers) can be serious contributors to security vulnerabilities. (SWEBOK; Chapter 4, page 4-3, section 3.2)

The choice of programming language can have a large effect on the likelihood of vulnerabilities being introduced during coding; uncritical usage of C and C++ are questionable choices from a security viewpoint. (SWEBOK; Chapter 4, page 4-3, section 3.2, 4th paragraph)

Constructors/Programmers need knowledge of good practices and common vulnerabilities; a good list can be found at www.cwe.mitre.org. (SWEBOK; Chapter 4, page 4-4, section 3.5)

In addition to vulnerabilities resulting from requirements or design and those resulting from poor choice or use of construction languages, faults introduced during construction can become security vulnerabilities. This includes not only faults in security functionality but also faults elsewhere that allow bypassing of such functionality or other security weaknesses or violations. (SWEBOK; Chapter 4, page 4-4, section 3.5)

Code should have knowledgeable security-oriented review. (SWEBOK; Chapter 4, page 4-4, section 3.5, 6th bullet)

Useful automatic static analysis of code for security weaknesses is available for several common programming languages and should be used. (SWEBOK; Chapter 4, page 4-1, Introduction, 6th paragraph and page 4-4, section 3.5, 7th bullet)

Reused and off-the-shelf software should meet the same security requirements as new software and the same degree of uncertainty, although confidence can be achieved in a different manner. (SWEBOK; Chapter 4, page 4-4, section 3.4)

Software Testing

[[1] Chapter 5, [2] Section 29.6, [3] Section 24.3]

Security testing can be relevant at all levels of testing, but is particularly relevant at the levels where security properties become emergent properties. (SWEBOK; Chapter 5, page 5-3, section 2.1)

In practice, large scale random or fuzz testing has been useful – particularly for legacy software. (SWEBOK; Chapter 5, page 5-6, section 3.2.6)

Security testing includes testing of security functionality as well as attack or penetration testing. Objectives include the correctness or reliability of security functionality and the preservation of security properties (including security functionality not being bypassed). Testing is needed of secure (and safe) failure, secure continuation of service after attack or security violation, and secure recovery. (SWEBOK; Chapter 5, page 5-4, section 2.2 and Chapter 5, page 5-5, section 2.2.10)

Among the tests techniques based on the nature of the application are those for security-critical systems. (SWEBOK; Chapter 5, page 5-6, section 3.7)

As a test technique, penetration testing attempts to find and exploit security weaknesses and vulnerabilities. Currently it mainly relies on the expertise of those conducting it although attack patterns have been cataloged (www.capec.mitre.org) and limited tools exist in some areas such as web applications. (SWEBOK; Chapter 5, page 5-6, section 3.7)

Failures to penetrate and/or fixing the vulnerabilities found during penetration testing adds little or nothing to the grounds for confidence (unless large, varied, and expert penetration tests are performed – something few can afford or have time or capability for). (SWEBOK; Chapter 5, page 5-7, section 4.1)

Sensitive data should not be used in testing. (SWEBOK; Chapter 5, page 5-9, section 5.2.2)

Security testing should be done in a manner that does not endanger other entities or activities and include testing in an environment that mimics the operational environment. (SWEBOK; Chapter 5, page 5-10, section 5.2.4)

Among the kinds of errors or defects that can cause failure is insertion of security weaknesses or vulnerabilities (www.cwe.mitre.org) and the omission or misuse of security functionality. (SWEBOK; Chapter 5, page 5-10, section 5.2.7)

Software Maintenance

[[2] Section 19.4]

All the security considerations in requirements, design, construction, and testing (and possibly elsewhere) potentially apply to maintenance. Testing is needed of secure (and safe) failure, secure continuation of service after attack or security violation, security during stoppages, and secure recovery. (SWEBOK; Chapter 6, page 6-1, section 1.2)

Key security issues in maintenance include: (SWEBOK; Chapter 6, page 6-3, section 2 and/or page 6-7, section 3.2.1)

- Addressing new security threats and newly discovered kinds of software vulnerabilities.
- The handling of information regarding discovered vulnerabilities usually needs to be restricted especially information on how to attack using them.
- Updates or "patches" for security vulnerabilities often need to be released rapidly to avoid their (additional) exploitation.
- Maintaining the rationale and evidence for security achievement.

Security is normally difficult to add to legacy software. Among the steps one can, nevertheless, take are: (SWEBOK; Chapter 6, page 6-7, section 3.2.1 or page 6-9, section 4)

- Deprecate functionality that is dangerous or unneeded.
- Reduce attack surface (paths for attack).
- Reduce the privilege level of operating system mode or environment that it runs in.
- Surround legacy code with guards, wrapper, or sandbox.
- Perform security reviews on legacy code.
- Perform security-oriented static analysis on legacy code.
- Modify legacy code to make in closer to [security cognizant] standard for new code.

To be effective, reviews and static analysis need to be followed by taking corrective measures. (SWEBOK; Chapter 6, page 6-7, section 3.2.1 or page 6-9, section 4)

Software Configuration Management

[[2] 29.7]

Configuration management facilities generally need to be secure, even if the software they store does not have security requirements, e.g. in order to protect the intellectual property contained within that software from being stolen or damaged. (SWEBOK; Chapter 7, page 7-3, section 1.3, 1st paragraph and/or page 7-4, section 1.3.3)

Accountability (possibly including non-repudiation) needs to be established for items. (SWEBOK; Chapter 7, page 7-3, section 1.3.1)

In selecting tools include consideration of: (SWEBOK; Chapter 7, page 7-4, section 1.3.3)

- Access control.
- Accountability.

In acquiring software items, the origin and initial integrity of each item needs to be firmly established, and if possible, any existing documentation of grounds for confidence in its security properties obtained and placed under configuration control. (SWEBOK; Chapter 7, page 7-6, section 2.1.6). This can be particularly challenging for free and open source software (FOSS), whose origins can be obscure.

A released item with security requirements needs to have its integrity ensured, for example by a digital signature. (SWEBOK; Chapter 7, page 7-9, section 6.2)

Software Engineering Management

[[1] Chapter 7 particularly Section 7.5, [3] Section 30.2]

Software engineers must be assured that adequate capability and resources are available to ensure the project is successfully completed including resulting in adequate confidence in this success. (SWEBOK; Chapter 8, page 8-4, section 1.2)

In software project planning include: (SWEBOK; Chapter 8, page 8-4, section 2)

- Security management: security management is closely tied to risk management and quality management but can influence all aspects of project management and planning.
- Closure activities include destruction of sensitive information and software, and possibly the medium on which copies resided.
- Having a plan that if followed is highly likely to result in adequate security.
- Planning to obtain the evidence and identify its relationships to security requirements such as to provide grounds for the needed confidence in security.

Security-related risks should be considered together with all other risks. (SWEBOK; Chapter 8, page 8-5, section 2.5)

Software-related security measurement is mainly based on general quality measurement and improvement activities, as well as measurements used in software engineering process definition and improvement activities. Vulnerabilities are treated as a special class of fault, and, to a lesser extent, measures of verifiability are used. Measurements supplied by security-oriented static analysis tools are also commonly used. The activities and changes in activities needed to achieve and assure security are defined and the software process changed, institutionalized, improved, and managed. Both of these areas have associated sets of measurement methods that are also used for their security-related aspects. Additional measures can relate to threats and losses. (SWEBOK; Chapter 8, page 8-7, section 6.2, 3rd bullet)

Software Engineering Process

[[1]Sections 7.5, 7.6, and Chapter 8, [2] Section 18.2, [3] Section 30.2]

Process definition needs to address security properties of the product and its life cycle. For example, it needs to consider the security aspects of development including the security of the development environment and the possible maliciousness of persons involved in development. (SWEBOK; Chapter 9, page 9-2, section 2)

Modeling the software's security properties, particularly its design, can be used for design rationale and verification. (SWEBOK; Chapter 9, page 9-7, section 4.3.1 and/or Chapter 10, page 10-4, section 2.2, 3rd bullet)

Software Engineering Tools and Methods

[[1] Sections 3.5.1, 4.2, 5.2.2, and 5.5]

Security-oriented static analysis tools should be used when security is a concern. However, static analysis tools for correctness can be at least a partial substitute. (SWEBOK; Chapter 10, page 10-2, section 1.3)

Among the methods useful for security are: (SWEBOK; Chapter 10, page 10-4, section 2.1)

- Abuse cases.
- Establishing security requirements.
- Security-oriented code review.
- Risk analysis including security risks and architectural risk analysis.
- Penetration testing.
- Testing security functionality with normal functional testing techniques, and risk-based security testing based on attack patterns.
- Monitoring software behavior is a defensive technique. Plus, knowledge gained by understanding attacks and exploits should be cycled back into software development activities.

Formal methods are highly useful for high security systems. (SWEBOK; Chapter 10, page 10-4, section 2.2)

Software Quality

[[1] Chapter 2, [3] Chapter 26]

Software quality management processes must address the degree of uncertainty regarding achievement of security-related requirements. (SWEBOK; Chapter 11, page 11-3, section 2, 4th paragraph)

Reused and off-the-shelf software should meet the same security requirements as new software and the same degree of uncertainty, although confidence can be achieved in a different manner. Free and open source software should also meet the same security requirements, but can present particular challenges in achieving appropriate confidence levels. This should be considered when choosing to use such software in an application with security requirements. (SWEBOK; Chapter 11, page 11-4, section 2.1, 1st paragraph

Practical considerations influence factors include the domain of the system being security critical. (SWEBOK; Chapter 11, page 11-6, section 3.1.1)

ISO/IEC 25010 associates the following qualities with security: confidentiality, integrity, non-repudiation, accountability, authenticity, security compliance, and immunity (the degree to which the software product is resistant to attack), plus related survivability and safety. (SWEBOK; Chapter 11, page 11-6, section 3.1.2, 1st paragraph)

ISO/IEC13335-1 states security is, “All aspects related to defining, achieving, and maintaining confidentiality, integrity, availability, accountability, authenticity, and reliability.” (SWEBOK; Chapter 11, page 11-6, section 3.1.2, 1st paragraph)

Concern for security-related principles, poor practices, weaknesses, and vulnerabilities are integrated into traditional quality practices such as reviews and measurement. (SWEBOK; Chapter 11, page 11-6, section 3.3, 1st paragraph and/or page 11-3, section 2, 4th paragraph)

Related Disciplines of Software Engineering

[[2] Chapter 1]

Security engineering is a related discipline as is security management. Security engineering often is said to encompass information security and information assurance. Computer security subsumes software security. While ethics, law, and governance are relevant for other aspects of software engineering, they are particularly relevant to security.

References

[1] Allen, Julia H., Sean Barnum, Robert J. Ellison, Gary McGraw, Nancy R. Mead. *Software Security Engineering: A Guide for Project Managers*. Addison Wesley, 2008.

[2] Bishop, Matt. *Computer Security: Art and Practice*, Addison-Wesley, 2003.

[3] Summerville, Ian. *Software Engineering, 8th Edition.* Addison Wesley, 2006.

Further Reading

Redwine, Samuel T., Jr. (Editor). *Software Assurance: A Curriculum Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software Version 1.2.* US Department of Homeland Security, 2007. Available at https://buildsecurityin.us-cert.gov/daisy/bsi/940-BSI/version/default/part/AttachmentData/data/CurriculumGuideToTheCBK.pdf