Software security assurance

Software security assurance is a process that helps design and implement software that protects the data and resources contained in and controlled by that software. Software is itself a resource and thus must be afforded appropriate security.

Since the number of threats specifically targeting software is increasing, the security of our software that we produce or procure must be assured. "Dependence on information technology makes software assurance a key element of business continuity, national security, and homeland security."[1]

Contents

What is software security assurance?

Software Security Assurance (SSA) is the process of ensuring that software is designed to operate at a level of security that is consistent with the potential harm that could result from the loss, inaccuracy, alteration, unavailability, or misuse of the data and resources that it uses, controls, and protects.

The software security assurance process begins by identifying and categorizing the information that is to be contained in, or used by, the software. The information should be categorized according to its sensitivity. For example, in the lowest category, the impact of a security violation is minimal (i.e. the impact on the

software owner's mission, functions, or reputation is negligible). For a top category, however, the impact may pose a threat to human life; may have an irreparable impact on software owner's missions, functions, image, or reputation; or may result in the loss of significant assets or resources.

Once the information is categorized, security requirements can be developed. The security requirements should address access control, including network access and physical access; data management and data access; environmental controls (power, air conditioning, etc.) and off-line storage; human resource security; and audit trails and usage records.

What causes software security problems?

All security vulnerabilities in software are the result of security bugs, or defects, within the software. In most cases, these defects are created by two primary causes: (1) non-conformance, or a failure to satisfy requirements; and (2) an error or omission in the software requirements.

Non-conformance, or a failure to satisfy requirements

A non-conformance may be simple–the most common is a coding error or defect–or more complex (i.e., a subtle timing error or input validation error). The important point about non-conformance is that verification and validation techniques are designed to detect them and security assurance techniques are designed to prevent them. Improvements in these methods, through a software security assurance program, can improve the security of software.

Errors or omissions in software requirements

The most serious security problems with software-based systems are those that develop when the software requirements are incorrect, inappropriate, or incomplete for the system situation. Unfortunately, errors or omissions in requirements are more difficult to identify. For example, the software may perform exactly as required under normal use, but the requirements may not correctly deal with some system state. When the system enters this problem state, unexpected and undesirable behavior may result. This type of problem cannot be handled within the software discipline; it results from a failure of the system and software engineering processes which developed and allocated the system requirements to the software.

Software security assurance activities

There are two basic types of Software Security Assurance activities.

1. Some focus on ensuring that information processed by an information system is assigned a proper sensitivity category, and that the appropriate protection requirements have been developed and met in the system.
2. Others focus on ensuring the control and protection of the software, as well as that of the software support tools and data.

At a minimum, a software security assurance program should ensure that:

1. A security evaluation has been performed for the software.
2. Security requirements have been established for the software.
3. Security requirements have been established for the software development and/or operations and maintenance (O&M) processes.

4. Each software review, or audit, includes an evaluation of the security requirements.
5. A configuration management and corrective action process is in place to provide security for the existing software and to ensure that any proposed changes do not inadvertently create security violations or vulnerabilities.
6. Physical security for the software is adequate.

Building in security

Improving the software development process and building better software are ways to improve software security, by producing software with fewer defects and vulnerabilities. A first-order approach is to identify the critical software components that control security-related functions and pay special attention to them throughout the development and testing process. This approach helps to focus scarce security resources on the most critical areas.

Tools and techniques

There are many commercial off-the-shelf (COTS) software packages that are available to support software security assurance activities. However, before they are used, these tools must be carefully evaluated and their effectiveness must be assured.

Common weaknesses enumeration

One way to improve software security is to gain a better understanding of the most common weaknesses that can affect software security. With that in mind, there is a current community-based program called the Common Weaknesses Enumeration project,[2] which is sponsored by The Mitre Corporation to identify and describe such weaknesses. The list, which is currently in a very preliminary form, contains descriptions of common software weaknesses, faults, and flaws.

Security architecture/design analysis

Security architecture/design analysis verifies that the software design correctly implements security requirements. Generally speaking, there are four basic techniques that are used for security architecture/design analysis. [3][4]

Logic analysis

Logic analysis evaluates the equations, algorithms, and control logic of the software design.

Data analysis

Data analysis evaluates the description and intended usage of each data item used in design of the software component. The use of interrupts and their effect on data should receive special attention to ensure interrupt handling routines do not alter critical data used by other routines.

Interface analysis

Interface analysis verifies the proper design of a software component's interfaces with other components of the system, including computer hardware, software, and end-users.

Constraint analysis

Constraint analysis evaluates the design of a software component against restrictions imposed by requirements and real-world limitations. The design must be responsive to all known or anticipated restrictions on the software component. These restrictions may include timing, sizing, and throughput constraints, input and output data limitations, equation and algorithm limitations, and other design limitations.

Secure code reviews, inspections, and walkthroughs

Code analysis verifies that the software source code is written correctly, implements the desired design, and does not violate any security requirements. Generally speaking, the techniques used in the performance of code analysis mirror those used in design analysis.

Secure Code reviews are conducted during and at the end of the development phase to determine whether established security requirements, security design concepts, and security-related specifications have been satisfied. These reviews typically consist of the presentation of material to a review group. Secure code reviews are most effective when conducted by personnel who have not been directly involved in the development of the software being reviewed.

Informal reviews

Informal secure code reviews can be conducted on an as-needed basis. To conduct an informal review, the developer simply selects one or more reviewer(s) and provides and/or presents the material to be reviewed. The material may be as informal as pseudo-code or hand-written documentation.

Formal reviews

Formal secure code reviews are conducted at the end of the development phase for each software component. The client of the software appoints the formal review group, who may make or affect a "go/no-go" decision to proceed to the next step of the software development life cycle.

Inspections and walkthroughs

A secure code inspection or walkthrough is a detailed examination of a product on a step-by-step or line-by-line (of source code) basis. The purpose of conducting secure code inspections or walkthroughs is to find errors. Typically, the group that does an inspection or walkthrough is composed of peers from development, security engineering and quality assurance.

Security testing

Software security testing, which includes penetration testing, confirms the results of design and code analysis, investigates software behaviour, and verifies that the software complies with security requirements. Special security testing, conducted in accordance with a security test plan and procedures, establishes the compliance of the software with the security requirements. Security testing focuses on

locating software weaknesses and identifying extreme or unexpected situations that could cause the software to fail in ways that would cause a violation of security requirements. Security testing efforts are often limited to the software requirements that are classified as "critical" security items.

See also

- Secure by design
- Computer security
- Security engineering
- Software protection

References

1. Karen Mercedes, Theodore Winograd "Enhancing The Development Life Cycle To Produce Secure Software" (http://www.thedacs.com/techs/abstract/358844) Archived (https://web.archive.org/web/20120330161131/http://www.thedacs.com/techs/abstract/358844) 2012-03-30 at the Wayback Machine, *Data & Analysis Center for Software*, October 2008
2. "Common Weaknesses Enumeration Project" (http://www.cve.mitre.org/cwe/index.html). Retrieved 26 August 2010.
3. Web Application Security Testing (https://pentesting.company/web-application-security-testing/)
4. "A Catalog of Security Architecture Weaknesses". 2017 IEEE International Conference on Software Architecture (ICSA). doi:10.1109/ICSAW.2017.25.