**Chapter 20: Building Systems with Assurance**

* Designing systems with assurance requires appropriate assurance levels at every step .
* The chapter emphasizes documentation and methods to support assurance claims and provides context for formal program verification and testing methodologies .

**20.1 Assurance in Requirements Definition and Analysis**

* Understanding requirements involves ensuring completeness and correctness within security policy models .
* The iterative process starts with threat definition, leading to detailed requirements used in system design and maintenance .

**20.1.1 Threats and Security Objectives**

* Identifying threats and developing security objectives are crucial for building secure systems .
* A threat is a potential occurrence with an undesirable effect on system assets or resources, while a vulnerability is a weakness that enables a threat .
* Security threats include breaches of confidentiality, disruptions of integrity, or denials of service, originating from both outside and inside the system .

**20.1.2 Architectural Considerations**

* Early architectural decisions involve determining the primary focus of control for security enforcement mechanisms, which can be user identity or operations .
* Centralizing security functions versus distributing them among system components is another key architectural decision .
* Centralized mechanisms are easier to analyze for assurance, while distributed mechanisms may offer better performance .

**20.1.2.1 Security Mechanisms and Layered Architecture**

* Computer architectures are layered, with security enforcement mechanisms residing at any layer .
* Layers include application, services/middleware, operating system, and hardware, each processing requests and passing them to the next layer .
* Selecting the correct layer for a mechanism is crucial for efficiency and effectiveness, considering protection of lower layers .

**20.1.2.2 Building Security In or Adding Security Later**

* Integrating security from the beginning is essential, as adding it later may not achieve high trust due to the lack of fundamental concepts .
* A reference monitor, implemented by a reference validation mechanism (RVM), mediates all access to objects by subjects and must be tamper-proof and always invoked .
* The trusted computing base (TCB) consists of all protection mechanisms responsible for enforcing a security policy .

**20.1.3 Policy Definition and Requirements Specification**

* A specification describes the characteristics of a computer system or program, with a security specification detailing desired security properties .
* Good specifications are clear, unambiguous, and complete, written at different levels of abstraction .
* Methods for defining policies include extracting requirements from standards like the Common Criteria, combining threat analysis with existing policies, or mapping the system to an existing model .

**20.1.4 Justifying Requirements**

* Defined policies must be shown to be complete and consistent .
* The ITSEC's security target (ST) defines security threats and functional requirements, with a suitability analysis justifying the functional requirements meet the threats .
* The suitability analysis maps threats to requirements and assumptions, providing an informal basis for asserting the threat is adequately handled if assumptions hold and requirements are met .

**20.2 Assurance during System and Software Design**

* Design assurance establishes that the system design enforces security requirements .
* Techniques include a specification of the requirements, a specification of the system design, and processes for examining how well the design meets the requirements .
* Modularity and layering simplify the system, making it more amenable to security analysis .

**20.2.1 Design Techniques That Support Assurance**

* Large systems can be broken into layers, simplifying specification development at different abstraction levels .
* Subsystems, components, subcomponents, and modules are different levels of a system .
* Design principles such as least privilege support security, with modular structure in design .

**20.2.2 Design Document Contents**

* Design documentation is crucial, requiring rigorous specifications for security requirements .
* Specifications can be informal, semiformal, or formal, with more formal specifications enabling rigorous security analysis .
* Documentation must specify security functions, external interfaces, and internal design .

**20.2.2.1 Security Functions Summary Specification**

* The security functions summary specification identifies high-level security functions defined for the system .
* It should include descriptions of individual security functions, an overview of how they work together, and a mapping to security requirements .

**20.2.2.2 External Functional Specification**

* The external functional specification describes external interfaces to a system, component, subcomponent, or module, detailing parameters, effects, exceptions, and error conditions .
* Technical content includes a component overview, data descriptions, and interface descriptions .
* Interface descriptions follow a standard syntax, identifying input and output parameters, exception conditions, effects, and error messages .

**20.2.2.3 Internal Design Description**

* An internal design description details the internal structures and functions of the system's components .
* High-level design documents focus on subsystems or components, addressing their structures, functions, and uses .
* Technical content includes an overview of the parent component, a detailed description of the component, and the security relevance of the component .

**20.2.2.4 Internal Design Specification**

* Internal design specification is useful for specifying the lowest layer of decomposition and the modules that make up that layer .
* Contains elements of the high-level design of the component and the low-level design of each module in the component .

**20.2.3 Building Documentation and Specification Considerations**

* Time, cost, and efficiency issues may impact how a development organization creates a complete set of documents .
* Modification specifications describe changes in existing modules, functions, or components, the addition of new modules, functions, or components .
* Supplemental specification created to describe missing security functionality .

**20.2.3.3 Formal Specifications**

* Requirements specifications, security functions summary specifications, functional specifications, and design specifications can be informal, semiformal, or formal .
* Formal specifications are written in formal languages based on well-defined syntax and sound semantics .

**20.2.4 Justifying That Design Meets Requirements**

* The nature of the specification limits the techniques that can validate the specified design .
* Requirements tracing, informal correspondence, and informal arguments can be used to analyzed a informal specification .
* Formal methods, such as formal specifications and precise mathematical proofs of correctness, produce higher assurance .

**20.2.4.1 Requirements Tracing and Informal Correspondence**

* Requirements tracing identifies specific security requirements met by parts of a specification .
* Informal correspondence shows that a specification is consistent with an adjacent level of specification .
* Together, these methods can provide confidence that specifications constitute a complete and consistent implementation of the security requirements .

**20.2.4.2 Informal Arguments**

* Informal arguments use an approach similar to mathematical proofs to analyze the completeness and correctness of security objectives .
* Protection profiles define threats and security objectives, with the rationale section justifying the objectives are adequate to prevent the threats .

**20.2.4.3 Formal Methods: Proof Techniques**

* The specifiers usually intend to process the specification using an automated tool such as a proof-based technology or a model checker .
* Formal proof mechanisms are general-purpose techniques that are usually based on logic such as the predicate calculus .
* A model checker is an automated tool with a specific security model and processes a specification to determine if the specification meets the constraints of the model .

**20.2.4.4 Review**

* A formal review process gains consensus on the appropriateness of assurance evidence .
* Review process has three critical parts: review guidelines, conflict resolution methods, and completion procedures .
* Managers may not be moderators, scribes, or observers and may be selected as reviewers only if their technical expertise is needed and cannot be provided otherwise .

**20.3 Assurance in Implementation and Integration**

* Testing shows that an implementation meets its security requirements .
* Implementation should be modular, with well-defined modules having a minimal number of well-defined interfaces .
* The choice of programming language affects implementation assurance with some languages strongly supporting security .

**20.3.1 Implementation Considerations That Support Assurance**

* Languages with features such as strong typing, built- in buffer overflow protections (such as array bounds handling), data hiding, modularity, domains and domain access protections, garbage collection, and error handling support the development of more secure, trustworthy, and reliable programs .
* Coding standards can compensate for some of the security enforcement limitations .

**20.3.2 Assurance through Implementation Management**

* Configuration management controls changes made in the system’s hardware, software, firmware, documentation, testing, test fixtures, and test documentation .
* The configuration management system is made up of several tools or manual processes and should perform several functions including version control and tracking and change authorization .
* Integration procedures and tools for product generation ensures that the system generation tools process properly authorized versions .

**20.3.3 Justifying That the Implementation Meets the Design**

* Code reviews, requirements tracing, informal correspondence, security testing, and formal proof techniques can be used to enhance assurance about the implementation .
* Comments in the code typically show the results of a requirement trace and a correspondence between the code and the lowest level of design documentation .

**20.3.3.1 Security Testing**

* Functional testing, sometimes called black box testing, is testing of an entity to determine how well it meets its specification .
* Structural testing, sometimes called white box testing, is testing based on an analysis of the code in order to develop test cases .
* Security testing consists of security functional testing, security structural testing and security requirements testing .

**20.3.3.2 Security Testing Using PGWG**

* PGWG, the PAT (Process Action Team) Guidance Working Group, presents a systematic approach to system and requirements test development using successive decomposition of the system and requirements tracing .
* Requirements are mapped to successively lower levels of design using test matrices .

**20.3.3.2.1 Test Matrices**

* The PGWG methodology defines two levels of test matrices (high and low) .
* The rows of the high-level matrix are the entity subsystems or major components .
* At the lowest level, matrix rows are the interfaces to the subsystem or component .

**20.3.3.2.2 Test Assertions**

* Test assertions are created by reviewing design documentation and identifying conditions that are security-relevant, testable, and analyzable .
* Assertions are at a very fine level of granularity, and each assertion should generate one or more individual tests that illustrate that the assertion is met .

**20.3.3.2.3 Test Specifications**

* Test specifications describe specific tests required to meet the assertions .
* PGWG suggests the use of high-level test specifications (HLTS) to describe and specify the test cases for each interface, and low-level test specifications (LLTS), which provide specific information about each test case, such as setup conditions, cleanup conditions, and other environmental conditions .

**20.3.3.3 Formal Methods: Proving That Programs Are Correct**

* Techniques exist for proving properties about programs, used during the coding process to help avoid bugs .
* This works best on small parts of a program that perform a well-defined task .

**20.4 Assurance during Operation and Maintenance**

* While a system is in operation, bugs will occur, requiring maintenance on the system .
* The action taken for a maintenance release or bug fix should follow the same security procedures used during the original development .

**20.5 Summary**

* Security assurance is an integral part of the life cycle of product or system development .
* Assurance measures are taken at every step of the process, from requirements development through design and development to testing and release, and must be supported during product or system operation .

**20.6 Research Issues**

* Creating systems and products from commercial off-the-shelf (COTS) components and providing as high a level of assurance as possible that the resulting system meets its requirements is an important issue .
* Adding appropriate assurance measures at appropriate times in the soft- ware engineering life cycle is another critical issue .
* Testing of systems and products for security is another area of active research .

**20.7 Further Reading**

* Yen and Paul [2052] present a short survey of six areas in which high assurance is critical .
* Early methodology for assertion-based testing and requirements correspondence in security is discussed by Bullough, Loomis, and Weiss [312] .
* Smith [1769] discusses the cost-benefit impacts of using formal methods for software assurance .