APPENDIX F

DESIGN PRINCIPLES FOR SECURITY

PROVIDING THE FOUNDATION FOR SYSTEMS SECURITY ENGINEERING[44]

Security design principles and concepts serve as the foundation for engineering trustworthy secure systems, including their constituent subsystems and components. These principles and concepts represent research, development, and application experience starting with the early incorporation of security mechanisms for trusted operating systems, to today's wide variety of fully networked, distributed, mobile, and virtual computing components, environments, and systems. The principles and concepts are intended to be universally applicable across this broad range of systems, as well as new systems as they emerge and mature.

The threat to be addressed is pervasive and can impact the trustworthiness of a system at any point during its life cycle. The principles are of interest to system developers who wish to mitigate the threat of insiders attempting to subvert systems at the hardware or software levels. Given the ubiquitous and increasing reliance on computing platforms and infrastructures to provide and enable mission and business capabilities, as well as data and information access and sharing, a shift toward robust, principles-based systems security engineering is both timely and relevant.

The security design principles are organized in a taxonomy that includes: *Security Architecture and Design* (i.e., organization, structure, interconnections, and interfaces); *Security Capability and Intrinsic Behaviors* (i.e., what the protections are and how they are provided); and *Life Cycle Security* (i.e., security process definition, conduct, and management). Application of these principles is intended to permit a demonstration of system trustworthiness through assurance based on reasoning about relevant and credible evidence. By applying the principles at different levels of abstraction (e.g., component design and composition), a sound security architecture based on trustworthy building blocks and a constructive approach can be developed. Definitions, underlying concepts, and other factors relevant to each principle and its application are also provided.

The security design principles and concepts presented in this appendix are intended to provide a basis for reasoning about a component or system. As reasoning tools, the inherent suitability of the principles and concepts in a particular situation will depend upon the practitioner's judgment. At times, the principles may be in conflict and their method of application may require tailoring. Within the overall system development process, the applicability of a particular principle may change due to evolving stakeholder requirements, protection needs, or constraints; architecture and design decisions and trade-offs; or by changes in risk tolerance. The security design principles and concepts should be an integral part of the total system solution. Their application should be planned for, scoped, and revisited throughout the engineering effort. Failure to properly apply these design principles and concepts may incur developmental, operational, or sustainment-driven risk.

[44] NIST acknowledges and appreciates the contribution of the U.S. Naval Postgraduate School (NPS) and the NPS Center for Information Systems Security Studies and Research (CISR) including principal investigators Paul Clark, Cynthia Irvine, and Thuy Nguyen, in providing the content for this appendix.

Table F-1 summarizes the taxonomy of security design principles. Each will be described in subsequent sections.

TABLE F-1: TAXONOMY OF SECURITY DESIGN PRINCIPLES

SECURITY DESIGN PRINCIPLES	
Security Architecture and Design	
Clear Abstractions	Hierarchical Trust
Least Common Mechanism	Inverse Modification Threshold
Modularity and Layering	Hierarchical Protection
Partially Ordered Dependencies	Minimized Security Elements
Efficiently Mediated Access	Least Privilege
Minimized Sharing	Predicate Permission
Reduced Complexity	Self-Reliant Trustworthiness
Secure Evolvability	Secure Distributed Composition
Trusted Components	Trusted Communication Channels
Security Capability and Intrinsic Behaviors	
Continuous Protection	Secure Failure and Recovery
Secure Metadata Management	Economic Security
Self-Analysis	Performance Security
Accountability and Traceability	Human Factored Security
Secure Defaults	Acceptable Security
Life Cycle Security	
Repeatable and Documented Procedures	Secure System Modification
Procedural Rigor	Sufficient Documentation

F.1 SECURITY ARCHITECTURE AND DESIGN

The following *structural design principles* affect the fundamental architecture of the system. This includes how the *system* is decomposed into its constituent *system elements*; and how the system elements relate to each other and the nature of the interfaces between elements.

F.1.1 Clear Abstractions

The principle of *clear abstractions* states that a system should have simple, well-defined interfaces and functions that provide a consistent and intuitive view of the data and how it is managed. The elegance (e.g., clarity, simplicity, necessity, sufficiency) of the system interfaces, combined with a precise definition of their functional behavior promotes ease of analysis, inspection, and testing as well as the correct and secure use of the system. The clarity of an abstraction is subjective. Examples reflecting application of this principle include avoidance of redundant, unused interfaces; *information hiding*; and avoidance of semantic overloading of interfaces or their parameters (e.g., not using one function to provide different functionality, depending on how it is used). Information hiding, also called *representation-independent programming*, is a design discipline to ensure that the internal representation of information in one system component is not visible to another system component invoking or calling the first

component, such that the published abstraction is not influenced by how the data may be managed internally.

F.1.2 Least Common Mechanism

The principle of *least common mechanism* states that the amount of mechanism common to more than one user and depended on by all users should be minimized [Popek74]. This implies that different components of a system should refrain from using the same mechanism to access a system resource. According to [Saltzer75], "Every shared mechanism (especially one involving shared variables) represents a potential information path between users and must be designed with great care to be sure it does not unintentionally compromise security." Implementing the principle of least common mechanism helps to reduce the adverse consequences of sharing system state among different programs. A single program corrupting a shared state (including shared variables) has the potential to corrupt other programs that are dependent on the state. The principle of least common mechanism also supports simplicity of design and addresses the issue of covert storage channels [Lampson73].

F.1.3 Modularity and Layering

The principles of *modularity* and *layering* are fundamental across system engineering disciplines. Modularity and layering derived from functional decomposition are effective in managing system complexity, by making it possible to comprehend the structure of the system. Yet, good modular decomposition, or refinement in system design is challenging and resists general statements of principle.

Modularity serves to isolate functions and related data structures into well-defined logical units. Layering allows the relationships of these units to be better understood, so that dependencies are clear and undesired complexity can be avoided. The security design principle of modularity extends functional modularity to include considerations based on trust, trustworthiness, privilege, and security policy. Security-informed modular decomposition includes the following: allocation of policies to systems in a network; allocation of system policies to layers; separation of system applications into processes with distinct address spaces; and separation of processes into subjects with distinct privileges based on hardware-supported privilege domains. The security design principles of modularity and layering are not the same as the concept of *defense in depth*, which is discussed in Section F.4.

F.1.4 Partially Ordered Dependencies

The principle of *partially ordered dependencies* states that the calling, synchronization, and other dependencies in the system should be partially ordered. A fundamental concept in system design is layering, whereby the system is organized into well-defined, functionally related modules or components. The layers are linearly ordered with respect to inter-layer dependencies, such that higher layers are dependent on lower layers. While providing functionality to higher layers, some layers can be self-contained and not dependent upon lower layers. While a partial ordering of all functions in a given system may not be possible, if circular dependencies are constrained to occur within layers, the inherent problems of circularity can be more easily managed. Partially ordered dependencies and system layering contribute significantly to the simplicity and coherency of the system design. Partially ordered dependencies also facilitate system testing and analysis.

F.1.5 Efficiently Mediated Access

The principle of *efficiently mediated access* states that policy-enforcement mechanisms should utilize the least common mechanism available while satisfying stakeholder requirements within

expressed constraints. The mediation of access to system resources (i.e., CPU, memory, devices, communication ports, services, infrastructure, data and information) is often the predominant security function of secure systems. It also enables the realization of protections for the capability provided to stakeholders by the system. Mediation of resource access can result in performance bottlenecks if the system is not designed correctly. For example, by using hardware mechanisms, efficiently mediated access can be achieved. Once access to a low-level resource such as memory has been obtained, hardware protection mechanisms can ensure that out-of-bounds access does not occur.

F.1.6 Minimized Sharing

The principle of *minimized sharing* states that no computer resource should be shared between system components (e.g., subjects, processes, functions) unless it is absolutely necessary to do so. Minimized sharing helps to simplify design and implementation. In order to protect user-domain resources from arbitrary active entities, no resource should be shared unless that sharing has been explicitly requested and granted. The need for resource sharing can be motivated by the principle of least common mechanism in the case internal entities, or driven by stakeholder requirements. However, internal sharing must be carefully designed to avoid performance and covert storage- and timing-channel problems. Sharing via common mechanism can increase the susceptibility of data and information to unauthorized access, disclosure, use, or modification and can adversely affect the inherent capability provided by the system. To help minimize the sharing induced by common mechanisms, such mechanisms can be designed to be reentrant or virtualized to preserve separation. Moreover, use of global data to share information should be carefully scrutinized. The lack of encapsulation may obfuscate relationships among the sharing entities.

F.1.7 Reduced Complexity

The principle of *reduced complexity* states that the system design should be as simple and small as possible. A small and simple design will be more understandable, more analyzable, and less prone to error. This principle applies to any aspect of a system, but it has particular importance for security due to the various analyses performed to obtain evidence about the emergent security property of the system. For such analyses to be successful, a small and simple design is essential. Application of the principle of reduced complexity contributes to the ability of system developers to understand the correctness and completeness of system security functions. It also facilitates identification of potential vulnerabilities. The corollary of reduced complexity states that the simplicity of the system is directly related to the number of vulnerabilities it will contain—that is, simpler systems contain fewer vulnerabilities. An important benefit of reduced complexity is that it is easier to understand whether the intended security policy has been captured in the system design, and that fewer vulnerabilities are likely to be introduced during engineering development. An additional benefit is that any such conclusion about correctness, completeness, and existence of vulnerabilities can be reached with a higher degree of assurance in contrast to conclusions reached in situations where the system design is inherently more complex.

F.1.8 Secure Evolvability

The principle of *secure evolvability* states that a system should be developed to facilitate the maintenance of its security properties when there are changes to its functionality structure, interfaces, and interconnections (i.e., system architecture) or its functionality configuration (i.e., security policy enforcement). These changes may include for example: new, enhanced, and upgraded system capability; maintenance and sustainment activities; and reconfiguration. Although it is not possible to plan for every aspect of system evolution, system upgrades and changes can be anticipated by analyses of mission or business strategic direction; anticipated

changes in the threat environment; and anticipated maintenance and sustainment needs. It is unrealistic to expect that complex systems will remain secure in contexts not envisioned during development, whether such contexts are related to the operational environment or to usage. A system may be secure in some new contexts, but there is no guarantee that its emergent behavior will always be secure. It is easier to build trustworthiness into a system from the outset, and it follows that the sustainment of system trustworthiness requires planning for change as opposed to adapting in an ad hoc or non-methodical manner. The benefits of this principle include reduced vendor life-cycle costs; reduced cost of ownership; improved system security; more effective management of security risk; and less risk uncertainty.

F.1.9 Trusted Components

The principle of *trusted components* states that a component must be trustworthy to at least a level commensurate with the security dependencies it supports (i.e., how much it is trusted to perform its security functions by other components). This principle enables the composition of components such that trustworthiness is not inadvertently diminished and where consequently the trust is not misplaced. Ultimately this principle demands some metric by which the trust in a component and the trustworthiness of a component can be measured on the same abstract scale. This principle is particularly relevant when considering systems and components in which there are complex chains of trust dependencies.[45] The principle also applies to a compound component that consists of several subcomponents (e.g., a subsystem), which may have varying levels of trustworthiness. The conservative assumption is that the overall trustworthiness of a compound component is that of its least trustworthy subcomponent. It may be possible to provide a security engineering rationale that the trustworthiness of a particular compound component is greater than the conservative assumption; however, any such rationale should reflect logical reasoning based on a clear statement of the trustworthiness goals, and relevant and credible evidence.[46]

F.1.10 Hierarchical Trust

The principle of *hierarchical trust* for components builds on the principle of trusted components and states that the security dependencies in a system will form a partial ordering if they preserve the principle of trusted components. The partial ordering provides the basis for trustworthiness reasoning when composing a secure system from heterogeneously trustworthy components. To be able to analyze a system composed of heterogeneously trustworthy components for its overall trustworthiness, it is essential to eliminate circular dependencies with regard to trustworthiness. If a more trustworthy component located in a lower layer of the system were to depend upon a less trustworthy component in a higher layer, this would in effect, put the components in the same "less trustworthy" equivalence class per the principle of trusted components. Trust relationships, or chains of trust, have various manifestations. For example, the root certificate of a certificate hierarchy is the most trusted node in the hierarchy, whereas the leaves in the hierarchy may be the least trustworthy nodes. Another example occurs in a layered high-assurance system where the security kernel (including the hardware base), which is located at the lowest layer of the system, is the most trustworthy component. This principle, however, does not prohibit the use of overly trustworthy components. There may be cases in a system of low trustworthiness, where it is reasonable to employ a highly trustworthy component rather than one that is less trustworthy (e.g., due to availability or other cost-benefit driver). For such a case, any dependency of the

[45] A trust dependency is also referred to as a *trust relationship* and there may be chains of trust relationships.

[46] The trustworthiness of a compound component is not the same as increased application of *defense–in-depth* layering within the component, or replication of components. Defense in depth techniques do not increase the trustworthiness of the whole above that of the least trustworthy component.

highly trustworthy component upon a less trustworthy component does not degrade the overall trustworthiness of the resulting low-trust system.

F.1.11 Inverse Modification Threshold

The principle of *inverse modification threshold* builds on the principle of trusted components and the principle of hierarchical trust, and states that the degree of protection provided to a component must be commensurate with its trustworthiness. As the trust placed in a component increases, the protection against unauthorized modification of the component should also increase to the same degree. This protection can come in the form of the component's own self-protection and innate trustworthiness, or from protections afforded to the component from other elements or attributes of the architecture (to include protections in the environment of operation).

F.1.12 Hierarchical Protection

The principle of *hierarchical protection* states that a component need not be protected from more trustworthy components. In the degenerate case of the most trusted component, it must protect itself from all other components. For example, if an operating system kernel is deemed the most trustworthy component in a system, then it must protect itself from all untrusted applications it supports, but the applications, conversely, do not need to protect themselves from the kernel. The trustworthiness of users is a consideration for applying the principle of hierarchical protection. A trusted computer system need not protect itself from an equally trustworthy user, reflecting use of untrusted systems in "system high" environments where users are highly trustworthy and where other protections are put in place to bound and protect the "system high" execution environment.

F.1.13 Minimized Security Elements

The principle of *minimized security elements* states that the system should not have extraneous trusted components. This principle has two aspects: the overall cost of security analysis and the complexity of security analysis. Trusted components, necessarily being trustworthy, are generally costlier to construct, owing to increased rigor of development processes. They also require greater security analysis to qualify their trustworthiness. Thus, to reduce the cost and decrease the complexity of the security analysis, a system should contain as few trustworthy components as possible. The analysis of the interaction of trusted components with other components of the system is one of the most important aspects of the verification of system security. If these interactions are unnecessarily complex, the security of the system will also be more difficult to ascertain than one whose internal trust relationships are simple and elegantly constructed. In general, fewer trusted components will result in fewer internal trust relationships and a simpler system.

F.1.14 Least Privilege

The principle of *least privilege* states that each component should be allocated sufficient privileges to accomplish its specified functions, but no more. This limits the scope of the component's actions, which has two desirable effects: the security impact of a failure, corruption, or misuse of the component will have a minimized security impact; and the security analysis of the component will be simplified. Least privilege is a pervasive principle that is reflected in all aspects of the secure system design. Interfaces used to invoke component capability should be available to only certain subsets of the user population, and component design should support a sufficiently fine granularity of privilege decomposition. For example, in the case of an audit mechanism, there may be an interface for the audit manager, who configures the audit settings; an interface for the audit operator, who ensures that audit data is safely collected and stored; and,

finally, yet another interface for the audit reviewer, who has need only to view the audit data that has been collected but no need to perform operations on that data.

In addition to its manifestations at the system interface, least privilege can be used as a guiding principle for the internal structure of the system itself. One aspect of internal least privilege is to construct modules so that only the elements encapsulated by the module are directly operated upon by the functions within the module. Elements external to a module that may be affected by the module's operation are indirectly accessed through interaction (e.g., via a function call) with the module that contains those elements. Another aspect of internal least privilege is that the scope of a given module or component should include only those system elements that are necessary for its functionality, and that the modes (e.g., read, write) by which the elements are accessed should also be minimal.

F.1.15 Predicate Permission

The principle of *predicate permission*[47] states that system designers should consider requiring multiple authorized entities to provide consent before a highly critical operation or access to highly sensitive data, information, or resources is allowed to proceed. The division of privilege among multiple parties decreases the likelihood of abuse and provides the safeguard that no single accident, deception, or breach of trust is sufficient to enable an unrecoverable action that can lead to significantly damaging consequences. The design options for such a mechanism may require simultaneous action (e.g., the firing of a nuclear weapon requires two different authorized individuals to give the correct command within a small time window) or a sequence of operations where each successive action is enabled by some prior action, but no single individual is able to enable more than one action.

F.1.16 Self-Reliant Trustworthiness

The principle of *self-reliant trustworthiness* states that systems should minimize their reliance on other systems for their own trustworthiness. A system should be trustworthy by default with any connection to an external entity used to supplement its function. If a system were required to maintain a connection with another external entity in order to maintain its trustworthiness, then that system would be vulnerable to malicious and non-malicious threats that result in loss or degradation of that connection. The benefit to this principle is that the isolation of a system will make it less vulnerable to attack. A corollary to this principle relates to the ability of the system (or system element) to operate in isolation and then resynchronize with other components when it is rejoined with them.

F.1.17 Secure Distributed Composition

The principle of *secure distributed composition* states that the composition of distributed components that enforce the same security policy should result in a system that enforces that policy at least as well as the individual components do. Many of the design principles for secure systems deal with how components can or should interact. The need to create or enable capability from the composition of distributed components can magnify the relevancy of these principles. In particular, the translation of security policy from a stand-alone to a distributed system or a system-of-systems can have unexpected or emergent results. Communication protocols and distributed data consistency mechanisms help to ensure consistent policy enforcement across a distributed system. To ensure a system-wide level of assurance of correct policy enforcement, the security architecture of a distributed composite system must be thoroughly analyzed.

[47] [Saltzer75] originally named this the *separation of privilege*. It is also equivalent to separation of duty.

F.1.18 Trusted Communication Channels

The principle of *trusted communication channels* states that when composing a system where there is a potential threat to communications between components (i.e., the interconnections between components), each communication channel must be trustworthy to a level commensurate with the security dependencies it supports (i.e., how much it is trusted by other components to perform its security functions). Trusted communication channels are achieved by a combination of restricting access to the communication channel (to help ensure an acceptable match in the trustworthiness of the endpoints involved in the communication) and employing end-to-end protections for the data transmitted over the communication channel (to help protect against interception, modification, and to further increase the overall assurance of proper end-to-end communication).

F.2 SECURITY CAPABILITY AND INTRINSIC BEHAVIORS

Security capability and intrinsic behavior design principles describe protection behavior that must be specified, designed, and implemented to achieve the emergent system property of security. The principles are applicable at the system, subsystem, and component levels of abstraction, and in general, are largely reflected in the system security requirements.

F.2.1 Continuous Protection

The principle of *continuous protection* states that all components and data used to enforce the security policy must have uninterrupted protection that is consistent with the security policy and the security architecture assumptions. No assurances that the system can provide the specified confidentiality, integrity, availability, and privacy protections for its design capability can be made if there are gaps in the protection. More fundamentally, any assurances about the ability to secure a delivered capability require that data and information are continuously protected. That is, there are no time periods during which data and information are left unprotected while under control of the system (i.e., during the creation, storage, processing, or communication of the data and information, as well as during system initialization, execution, failure, interruption, and shutdown). Continuous protection requires adherence to the precepts of the *reference monitor concept* (i.e., every request is validated by the reference monitor, the reference monitor is able to protect itself from tampering, and sufficient assurance of the correctness and completeness of the mechanism can be ascertained from analysis and testing), and the *principle of secure failure and recovery* (i.e., preservation of a secure state during error, fault, failure, and successful attack; preservation of a secure state during recovery to normal, degraded, or alternative operational modes).

Continuous protection also applies to systems designed to operate in varying configurations including those that deliver full operational capability and other degraded-mode configurations that deliver partial operational capability. The continuous protection principle requires that changes to the system security policies be traceable to the operational need that drives the configuration and be verifiable (i.e., it must be possible to verify that the proposed changes will not put the system into an insecure state). Insufficient traceability and verification may lead to inconsistent states or protection discontinuities due to the complex or undecidable nature of the problem. The use of pre-verified configuration definitions that reflect the new security policy enables analysis to determine that a transition from old to new policies is essentially atomic, and that any residual effects from the old policy are guaranteed to not conflict with the new policy. The ability to demonstrate continuous protection is rooted in the clear articulation of life cycle protection needs as stakeholder security requirements.

F.2.2 Secure Metadata Management

The principle of *secure metadata management* states that metadata must be considered as "first class" objects with respect to security policy when the policy requires complete protection of information or it requires the security subsystem to be self-protecting. This principle is driven by the recognition that a system, subsystem, or component cannot achieve self-protection unless it protects the data it relies upon for correct execution. Data is generally not interpreted by the system that stores it. It may have semantic value (i.e., it comprises information) to users and programs that process the data. In contrast, metadata is information about data, such as a file name or the date when the file was created. Metadata is bound to the target data that it describes in a way that the system can interpret, but it need not be stored inside of or proximate to its target data. There may be metadata whose target is itself metadata (e.g., the sensitivity level of a file name), to include self-referential metadata.

The apparent secondary nature of metadata can lead to a neglect of its legitimate need for protection, resulting in violation of the security policy that includes the exfiltration of information in violation of security policy. A particular concern associated with insufficient protections for metadata is associated with multilevel secure (MLS) computing systems. MLS computing systems mediate access by a subject to an object based on their relative sensitivity levels. It follows that all subjects and objects in the scope of control of the MLS system must be directly labeled or indirectly attributed with sensitivity levels. The *corollary of labeled metadata* for MLS systems states that objects containing metadata must be labeled. As with the protection needs assessment for data, attention should be given to ensure that appropriate confidentiality and integrity protections are individually assessed, specified, and allocated to metadata, as would be done for mission, business, and system data.

F.2.3 Self-Analysis

The principle of *self-analysis* states that a component must be able to assess its internal state and functionality to a limited extent at various stages of execution, and that this self-analysis capability must be commensurate with the level of trustworthiness invested in the system. At the system level, self-analysis can be achieved via hierarchical trustworthiness assessments established in a bottom up fashion. In this approach, the lower-level components check for *data integrity* and correct functionality (to a limited extent) of higher-level components. For example, trusted boot sequences involve a trusted lower-level component attesting to the trustworthiness of the next higher-level components so that a transitive chain of trust can be established. At the root, a component attests to itself, which usually involves an axiomatic or environmentally enforced assumption about its integrity. These tests can be used to guard against externally induced errors, or internal malfunction or transient errors. By following this principle, some simple errors or malfunctions can be detected without allowing the effects of the error or malfunction to propagate outside the component. Further, the self-test can also be used to attest to the configuration of the component, detecting any potential conflicts in configuration with respect to the expected configuration.

F.2.4 Accountability and Traceability

The principle of *accountability and traceability* states that it must be possible to trace security-relevant actions (i.e., subject-object interactions) to the entity on whose behalf the action is being taken. This principle requires a trustworthy infrastructure that can record details about actions that affect system security (e.g., an audit subsystem). To do this, the system must not only be able to uniquely identify the entity on whose behalf the action is being carried out, but also record the relevant sequence of actions that are carried out. Further, the accountability policy must require

the audit trail itself be protected from unauthorized access and modification. The principle of least privilege aids in tracing the actions to particular entities, as it increases the granularity of accountability. Associating actions with system entities, and ultimately with users, and making the audit trail secure against unauthorized access and modifications provides non-repudiation, because once an action is recorded, it is not possible to change the audit trail. Another important function that accountability and traceability serves is in the routine and forensic analysis of events associated with the violation of security policy. Analysis of the audit logs may provide additional information that may be helpful in determining the path or component that allowed the violation of security policy, and the actions of individuals associated with the violation of security policy.

F.2.5 Secure Defaults

The principle of *secure defaults* states that the default configuration of a system (to include its constituent subsystems, components, and mechanisms) reflects a restrictive and conservative enforcement of security policy. The principle of secure defaults applies to the initial (i.e., default) configuration of a system as well as to the security engineering and design of access control and other security functions that should follow a "deny unless explicitly authorized" strategy. The initial configuration aspect of this principle requires that any "as shipped" configuration of a system, subsystem, or component should not aid in the violation of the security policy, and can prevent the system from operating in the default configuration for those cases where the security policy itself requires configuration by the operational user.

Restrictive defaults mean that the system will operate "as-shipped" with adequate self-protection, and is able to prevent security breaches before the intended security policy and system configuration is established. In cases where the protection provided by the "as-shipped" product is inadequate, the stakeholder must assess the risk of using it prior to establishing a secure initial state. Adherence to the principle of secure defaults guarantees a system is established in a secure state upon successfully completing initialization. Moreover, in situations where the system fails to complete initialization, either it will perform a requested operation using secure defaults or it will not perform the operation. Refer also to the principles of continuous protection and secure failure and recovery which parallel this principle to provide the ability to detect and recover from failure.

The security engineering approach to this principle states that security mechanisms should deny requests unless the request is found to be well-formed and consistent with the security policy. The insecure alternative is to allow a request unless it is shown to be inconsistent with the policy. In a large system, the conditions that must be satisfied to grant a request that is by default denied are often far more compact and complete than those that would need to be checked in order to deny a request that is by default granted.

F.2.6 Secure Failure and Recovery

The principle of *secure failure and recovery* states that neither a failure in a system function or mechanism nor any recovery action in response to failure should lead to a violation of security policy. This principle parallels the principle of continuous protection to ensure that a system is capable of detecting (within limits) actual and impending failure at any stage of its operation (i.e., initialization, normal operation, shutdown, and maintenance) and to take appropriate steps to ensure that security policies are not violated. In addition, when specified, the system is capable of recovering from impending or actual failure to resume normal, degraded, or alternative secure operation while ensuring that a secure state is maintained such that security policies are not violated.

Failure is a condition in which a component's behavior deviates from its specified or expected behavior for an explicitly documented input. Once a failed security function is detected, the system may reconfigure itself to circumvent the failed component, while maintaining security, and still provide all or part of the functionality of the original system, or completely shut itself down to prevent any (further) violation of security policies. For this to occur, the reconfiguration functions of the system should be designed to ensure continuous enforcement of security policy during the various phases of reconfiguration. Another technique that can be used to recover from failures is to perform a *rollback* to a secure state (which may be the initial state) and then either shutdown or replace the service or component that failed such that secure operation may resume. Failure of a component may or may not be detectable to the components using it. The principle of secure failure indicates that components should fail in a state that denies rather than grants access. For example, a nominally "atomic" operation interrupted before completion should not violate security policy and hence must be designed to handle interruption events by employing higher-level atomicity and rollback mechanisms (e.g., transactions). If a service is being used, its atomicity properties must be well-documented and characterized so that the component availing itself of that service can detect and handle interruption events appropriately. For example, a system should be designed to gracefully respond to disconnection and support resynchronization and data consistency after disconnection.

Failure protection strategies that employ replication of policy enforcement mechanisms, sometimes called *defense in depth*, can allow the system to continue in a secure state even when one mechanism has failed to protect the system. If the mechanisms are similar, however, the additional protection may be illusory, as the adversary can simply *attack in series*. Similarly, in a networked system, breaking the security on one system or service may enable an attacker to do the same on other similar replicated systems and services. By employing multiple protection mechanisms, whose features are significantly different, the possibility of attack replication or repetition can be reduced. Analyses should be conducted to weigh the costs and benefits of such redundancy techniques against increased resource usage and adverse effects on the overall system performance. Additional analyses should be conducted as the complexity of these mechanisms increases, as could be the case for dynamic behaviors. Increased complexity generally reduces trustworthiness. When a resource cannot be continuously protected, it is critical to detect and repair any security breaches before the resource is once again used in a secure context.

F.2.7 Economic Security

The principle of *economic security* states that security mechanisms should not be costlier than the potential damage that could occur from a security breach. This is the security-relevant form of the cost-benefit analyses used in risk management. The cost assumptions of this analysis will prevent the system designer from incorporating security mechanisms of greater strength than necessary, where strength of mechanism is proportional to cost. It also requires analysis of the benefits of assurance relative to the cost of that assurance in terms of the effort expended to obtain relevant and credible evidence, and to perform the analyses necessary to assess and draw trustworthiness and risk conclusions from the evidence.

F.2.8 Performance Security

The principle of *performance security* states that security mechanisms should be constructed so that they do not degrade system performance unnecessarily. Both stakeholder and system design requirements for performance and security must be precisely articulated and prioritized. For the system implementation to meet its design requirements and be found acceptable to stakeholders (i.e., validation against stakeholder requirements), the designers must adhere to the specified constraints that capability performance needs place on protection needs. The overall impact of

computationally intensive security services (e.g., cryptography) should be assessed and be demonstrated to pose no significant impact to higher-priority performance considerations or deemed to be providing an acceptable trade-off of performance for trustworthy protection. Trade-off considerations should include less computationally intensive security services unless they are unavailable or insufficient. The insufficiency of a security service is determined by functional capability and strength of mechanism. The strength of mechanism must be selected appropriately with respect to security requirements as well as performance-critical overhead issues (e.g., cryptographic key management) and an assessment of the capability of the threat.

The principle of performance security leads to the incorporation of features that help in the enforcement of security policy, but incur minimum overhead, such as low-level hardware mechanisms upon which higher-level services can be built. Such low-level mechanisms are usually very specific, have very limited functionality, and are heavily optimized for performance. For example, once access rights to a portion of memory is granted, many systems use hardware mechanisms to ensure that all further accesses involve the correct memory address and access mode. Application of this principle reinforces the need to design security into the system from the ground up, and to incorporate simple mechanisms at the lower layers that can be used as building blocks for higher-level mechanisms.

F.2.9 Human Factored Security

The principle of *human factored security* states that the user interface for security functions and supporting services should be intuitive, user friendly, and provide appropriate feedback for user actions that affect such policy and its enforcement. The mechanisms that enforce security policy should not be intrusive to the user and should be designed not to degrade user efficiency. They should also provide the user with meaningful, clear, and relevant feedback and warnings when insecure choices are being made. Particular attention must also be given to interfaces through which personnel responsible for system operation and administration configure and set up the security policies. Ideally, these personnel must be able to understand the impact of their choices. They must be able to configure systems before start-up and administer them during runtime, in both cases with confidence that their intent is correctly mapped to the system's mechanisms. Security services, functions, and mechanisms should not impede or unnecessarily complicate the intended use of the system. There is often a trade-off between system usability and the strictness necessitated for security policy enforcement. If security mechanisms are frustrating or difficult to use, then users may disable or avoid them, or use the mechanisms in ways inconsistent with the security requirements and protection needs the mechanisms were designed to satisfy.

F.2.10 Acceptable Security

The principle of *acceptable security* requires that the level of privacy and performance the system provides should be consistent with the users' expectations. The perception of personal privacy may affect user behavior, morale, and effectiveness. Based on the organizational privacy policy and the system design, users should be able to restrict their actions to protect their privacy. When systems fail to provide intuitive interfaces, or meet privacy and performance expectations, users may either choose to completely avoid the system or use it in ways that may be inefficient or even insecure.

F.3 LIFE CYCLE SECURITY

Several principles guide and inform a definition of the system life cycle that incorporates the security perspective necessary to achieve the initial and continuing security of the system. A

secure system life cycle contributes to system comprehensibility and maintainability, as well as system integrity.[48]

F.3.1 Repeatable and Documented Procedures

The principle of *repeatable and documented procedures* states that the techniques and methods employed to construct a system component should permit the same component to be completely and correctly reconstructed at a later time. Repeatable and documented procedures support the development of a component that is identical to the component created earlier that may be in widespread use. In the case of other system artifacts (e.g., documentation and testing results), repeatability supports consistency and ability to inspect the artifacts. Repeatable and documented procedures can be introduced at various stages within the system life cycle and can contribute to the ability to evaluate assurance claims for the system. Examples include systematic procedures for code development and review; procedures for configuration management of development tools and system artifacts; and procedures for system delivery.

F.3.2 Procedural Rigor

The principle of *procedural rigor* states that the rigor of a system life cycle process should be commensurate with its intended trustworthiness. Procedural rigor defines the scope, depth, and detail of the system life cycle procedures. These procedures contribute to the assurance that the system is correct and free of unintended functionality in several ways. First, they impose checks and balances on the life cycle process such that the introduction of unspecified functionality is prevented. Second, rigorous procedures applied to systems security engineering activities that produce specifications and other design documents contribute to the ability to understand the system as it has been built, rather than trusting that the component as implemented, is the authoritative (and potentially misleading) specification. Finally, modifications to an existing system component are easier when there are detailed specifications describing its current design, instead of studying source code or schematics to try to understand how it works. Procedural rigor helps to ensure that security functional and assurance requirements have been satisfied, and it contributes to a better-informed basis for the determination of trustworthiness and risk posture. Procedural rigor should always be commensurate with the degree of assurance desired for the system. If the required trustworthiness of the system is low, a high level of procedural rigor may add unnecessary cost, whereas when high trustworthiness is critical, the cost of high procedural rigor is merited.

F.3.3 Secure System Modification

The principle of *secure system modification* states that system modification must maintain system security with respect to the security requirements and risk tolerance of stakeholders. Upgrades or modifications to systems can transform a secure system into an insecure one. The procedures for system modification must ensure that, if the system is to maintain its trustworthiness, the same rigor that was applied to its initial development is applied to any changes. Because modifications can affect the ability of the system to maintain its secure state, a careful security analysis of the modification is needed prior to its implementation and deployment. This principle parallels the principle of secure evolvability.

F.3.4 Sufficient Documentation

The principle of *sufficient documentation* states that personnel with responsibility to interact with the system should be provided with adequate documentation and other information such that they

[48] [Myers80] provides examples of subversion throughout the system life cycle.

contribute to rather than detract from system security. Despite attempts to comply with principles such as human factored security and acceptable security, systems are inherently complex, and the design intent for the use of security mechanisms is not always intuitively obvious. Neither are the ramifications of their misuse or misconfiguration. Uninformed and insufficiently trained users can introduce new vulnerabilities due to errors of omission and commission. The ready availability of documentation and training can help to ensure a knowledgeable cadre of personnel, all of whom have a critical role in the achievement of principles such as continuous protection. Documentation must be written clearly and supported by appropriate training that provides security awareness and understanding of security-relevant responsibilities.

F.4 APPROACHES TO TRUSTWORTHY SECURE SYSTEM DEVELOPMENT

This section introduces three overarching strategies that may be applied in the development of trustworthy secure systems. These approaches may be used individually or in combination.

F.4.1 Reference Monitor Concept

The *reference monitor concept* provides an abstract security model of the necessary and sufficient properties that must be achieved by any system mechanism claiming to securely enforce access controls. The reference monitor concept does not refer to any particular policy to be enforced by a system, nor does it address any particular implementation. Instead, the intent of this concept is to help practitioners avoid *ad hoc* approaches to the development of security mechanisms intended to enforce critical policies and can also be used to provide assurance that the system has not been corrupted by an insider. The abstract instantiation of the reference monitor concept is an "ideal mechanism" characterized by three properties: the mechanism is tamper-proof (i.e., it is protected from modification so that it always is capable of enforcing the intended access control policy); the mechanism is always invoked (i.e., it cannot be bypassed so that every access to the resources it protects is mediated); and the mechanism can be subjected to analysis and testing to assure that it is correct (i.e., it is possible to validate that the mechanism faithfully enforces the intended security policy and that it is correctly implemented).

While abstract mechanisms can be ideal, actual systems are not. The reference monitor concept provides an "ideal" toward which system security engineers can strive in the basic design and implementation of the most critical components of their systems, given practical constraints and limitations. Those constraints and limitations translate to risk that is managed through analyses and decisions applied to the architecture and design of the particular reference monitor concept implementation, and subsequently to its integration into a broader system architecture for a component, subsystem, infrastructure, system, or system-of-systems. Therefore, although originally used to describe a monolithic system, a generalization of the reference monitor concept serves well as the fundamental basis for the design of individual security-relevant system elements, collections of elements, and for systems. The generalization also guides the activities that obtain evidence used to substantiate claims that trustworthiness objectives have been achieved and to support determinations of risk.

F.4.2 Defense in Depth

Defense in depth describes security architectures constructed through the application of multiple mechanisms to create a series of barriers to prevent, delay, or deter an attack by an adversary. The application of some security components in a defense in depth strategy may increase assurance, but there is no theoretical basis to assume that defense in depth alone could achieve a level of trustworthiness greater than that of the individual security components used. That is, a defense in

depth strategy is not a substitute for or equivalent to a sound security architecture and design that leverages a balanced application of security concepts and design principles.

F.4.3 Isolation

Two forms of *isolation* are available to system security engineers: logical isolation and physical isolation. The former requires the use of underlying trustworthy mechanisms to create isolated processing environments. These can be constructed so that resource sharing among environments is minimized. Their utility can be realized in situations in which virtualized environments are sufficient to satisfy computing requirements. In other situations, the isolation mechanism can be constructed to permit sharing of resources, but under the control and mediation of the underlying security mechanisms, thus avoiding blatant violations of security policy.

Researchers continue to demonstrate that isolation for processing environments can be extremely difficult to achieve, so stakeholders must determine the potential risk of incomplete isolation, the consequences of which can include covert channels and side channels. Another form of logical isolation can be realized within a process. Traditionally obtained through the use of hardware mechanisms, a hierarchy of protected privilege domains can be developed within a process. Course-grained hierarchical isolation, and consequently privilege domain separation, is in common use in many modern commercial systems, and separates the user domain from that of the operating system. More granular hierarchical isolation is less common today; however, examples exist in prior research-prototype and commercial systems.

Physical isolation involves separation of components, systems, and networks by hosting them on separate hardware. It may also include the use of specialized computing facilities and operational procedures to allow access to systems only by authorized personnel. In many situations, isolation objectives may be achieved by a combination of logical and physical isolation. Security architects and operational users must be cognizant of the co-dependencies between the logical and physical mechanisms and must ensure that their combination satisfies security and assurance objectives. A full discussion of isolation is beyond the scope of this appendix.

APPLICATION OF SECURITY DESIGN PRINCIPLES TO COMMERCIAL PRODUCTS

For commercial products (i.e., commercial components, commodity products, and systems) to be trustworthy commensurate with their criticality, those products should be designed such that the security design principles and concepts are selected and applied appropriately throughout their entire system life cycle. Each security design principle must be assessed for its relevance, applicability, and validity. The security design principles described in this appendix have been demonstrated by industry in past work and have previously been codified into national and international standards and guidance documents, including, for example, the Department of Defense *Trusted Computer System Evaluation Criteria (TCSEC)* and ISO/IEC 15408, *Common Criteria for Information Technology Security Evaluation*. Many commercial products have been developed and evaluated against specifications from those design, engineering, and security standards up to and including the highest levels of assurance (e.g., TCSEC Classes A1 and B3). These products represent worked examples and use cases of highly trustworthy components and systems that have been verified to be highly resistant to penetration from determined adversaries, and, in the case of TCSEC Class A1, distinguished by substantially dealing with the problem of subversion of security mechanisms. To merit the trust of consumers, commercial products must demonstrate in a manner that can be independently verified, that the security design principles articulated in this appendix have been applied to produce components and systems that are both sound and logically coherent with respect to security.