Secure by Design Principles

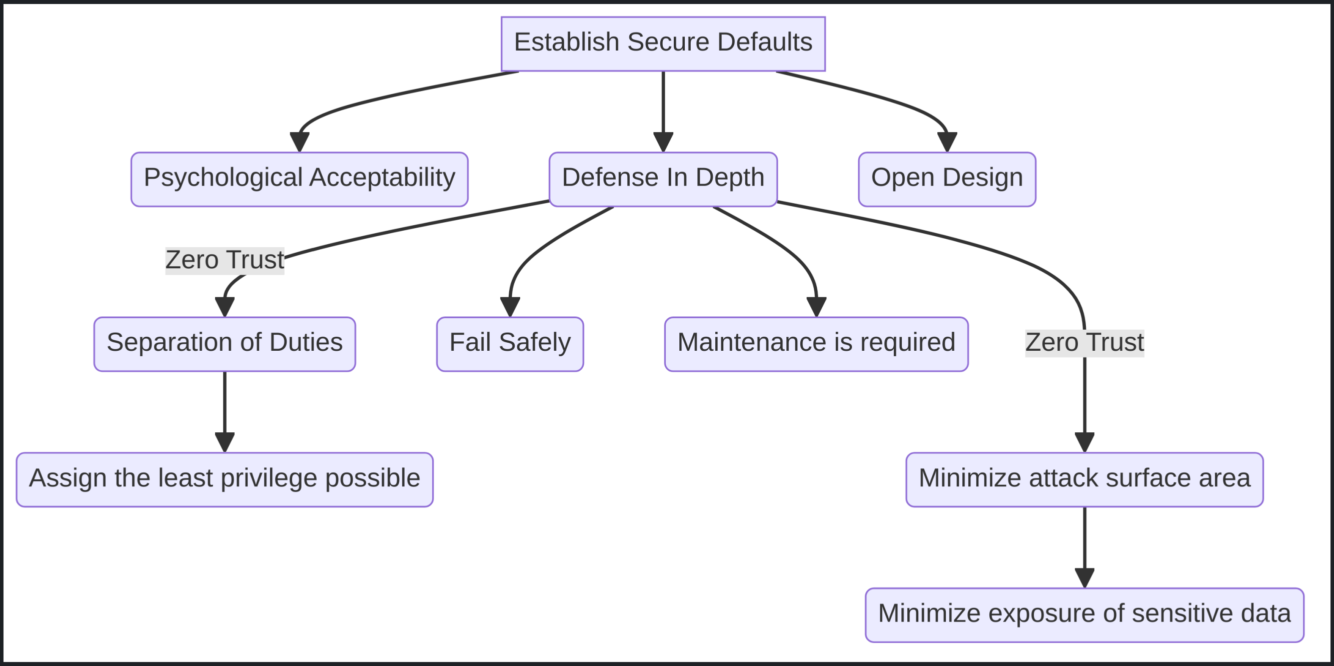
Secure By Design (SBD) uses design to drive security in software development. Several frameworks[1][2][3][4][5] and books[1][2] have been created. This approach is reinforced by Archer's Product Security Well-Tended Assumptions.

As described by the namesake book:

> Security should be the natural outcome of your development process. As applications increase in complexity, it becomes more important to bake security-mindedness into every step. The secure-by-design approach teaches best practices to implement essential software features using design as the primary driver for security.

SbD Principles

The implementation of SbD is often described in terms of design principles.


Establish Secure Defaults
Psychological Acceptability
Defense In Depth
Open Design
Zero Trust
Separation of Duties
Fail Safely
Maintenance is required
Zero Trust
Assign the least privilege possible
Minimize attack surface area
Minimize exposure of sensitive data

Design principles are not absolutes.

Establish secure defaults

The system should be secure by default on first use. Optional features should be disabled by default with explicit user actions required to enable them. Operations where permission is not explicitly granted should fail.

Closely related to the Principle of Least Surprise

It should not require an expert to make the system secure, it should require an expert to make it insecure

Psychological acceptability

Security mechanisms should not make the resource more difficult to access than if the security mechanisms were not present. Configuring and executing a program should be as easy and as intuitive as possible, and any output should be clear, direct, and useful[1]

This does not mean that security and assurance measures need to add no overhead or user exposed rigor to the system. The expectation that users secure their seat belt is appropriate, but the application of the seatbelt must not hinder the operation of the vehicle.

Users will most often take the path of least resistance

[1] "Design Principles," in "Principle of Psychological Acceptability" 348-349 - Bishop

Defense in depth

Single points of defense are avoided or mitigated by the incorporation of multiple layers of security safeguards and countermeasure.

Zero-Trust, part of a defense-in-depth strategy, acknowledges that all transactions between separate components of a system must be authenticated and authorized regardless of source or destination.

Complete Mediation, part of a defense-in-depth strategy, states that every time someone tries to access an object, the system should authenticate the privileges associated with that subject.

Assume any single protection can and will fail. Authenticate, authorize, encrypt and log accordingly.

Separation of Duties

No single role should have the ability to influence a system or process without checks and balances. This is related to but distinct from the concept of Least Privilege. Separation of duties is required to enable the ability to audit, verify and report on activity in a system.

Often called compartmentalization

Assign the least privilege possible

A user or process should have the minimum privileges, for the minimum amount of time, required for its function. Requirements should make it clear which resources are available for each user and process.

Limiting the blast radius in the case of exposure and exploitation

Maintenance is required

The “Software Development Lifecycle” (SDLC) often referenced must include both development and operation until retirement. Maintenance typically consumes 40-80 percent (60 percent average) of software costs[1].

Dependencies require updating for vulnerabilities and to remain on supported versions. Software at all levels of a deployed stack require regular security updates. A sane maintenance phase begins with contemplation in the design phase, and is partially the outcome of secure-by-design principles applied as a whole. As with failure, maintenance must be part of the plan from the beginning.

> Code is written once and read many times. Optimize for maintainability.

[1] Frequently Forgotten Fundamental Facts about Software Engineering - Glass

Fail safely

Failure paths are often less commonly accessed during normal operations and development. Attackers will trigger intentional failure within a system to expose internals and/or force it into an insecure state. Handling failure is an expected part of system operation.

Related to graceful degradation

> Triggering failure is a an expected method of attack.

Minimize attack surface area

Every feature and function of a system is a potential attack vector. Even security functionality can contain vulnerabilities and have a negative security impact. This applies conceptually to the technology stack in use as a whole, and to the attributes of components in that stack. There should be well understood and defined reasoning for increasing attack surface with the introduction of complexity or novelty.

This is often balanced against the concept of Least Common Mechanism: “*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.*”[1]

Related to the principle Economy of Mechanism and Keep It Simple Stupid (KISS)

> *Reusable safety components allow for more consistently predictable outcomes*

[1] The Protection of Information in Computer Systems, H. SALTZER & SCHROEDER

Minimize exposure of sensitive data

This means both minimizing the exposure of the company to the sensitive data storage of others, and being mindful of exposure of stored sensitive data to internal operations.

A data classification policy needs to be in place to determine what data is confidential and how to treat it.

> Data must be classified to consistently make reliable decisions about treatment

Open Design

System security must not rely on the secrecy of the implementation.   This is in contrast to “security by obscurity” wherein the security of the software is dependent upon the obscuring of the design itself.

This is a particularly important principle for cryptographic and authentication schemes. Well-designed cryptography implementations are published publicly. They’re peer-reviewed and interrogated by subject matter experts before being put into practice. This does not mean that all aspect of an operating system are published indiscriminately.   As an example from *Fail Safely,* raw exception messages should not surface for end users.

Review of the design itself must not result in the compromise of the safeguards in the system.