OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

INFORMATIK UND
MATHEMATIK

Secure Programming

Introduction

Prof. Dr. Christoph Skornia
christoph.skornia@hs-regensburg.de

OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

INFORMATIK UND
MATHEMATIK

SPG
K002

CHECKEN SIE EIN.
STOPPEN SIE DAS VIRUS.

Nutzen Sie die Corona-Warn-App! Scannen Sie den QR-Code und tragen Sie aktiv dazu bei, mögliche Infektionsketten schnell und effektiv zu durchbrechen.

Laden im App Store | JETZT BEI Google Play | CORONA WARN-APP | Die Bundesregierung

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG
INFORMATIK UND MATHEMATIK

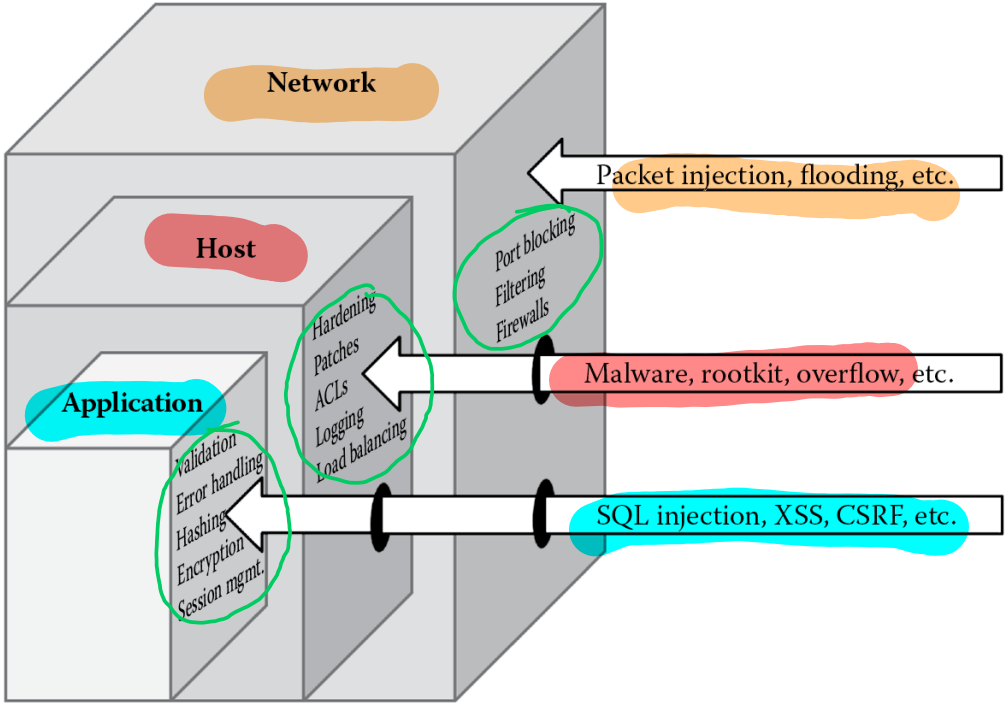
Agenda

1. Software Security Requirements
2. Vulnerabilities
3. Protections
   - ❑ Initialization
   - ❑ Input Validation
     - ❑ Overflow Attacks
     - ❑ Injection Attacks
   - ❑ Principle of Least Privilege
   - ❑ File and Memory Operations
4. Encryption
5. Authentication
6. Secure Software Development
   - Capability Maturity Model
   - Secure Software Development Life Cycle Model
   - Audit and Static Analysis

Playground of Information Security

general assumptions:

- every system, which contains information belongs to the playground of information security
- everything what can fail will fail (sooner or later)
- everything what can be attacked will be attacked
- 100% security is impossible but everything which is improving the status is good

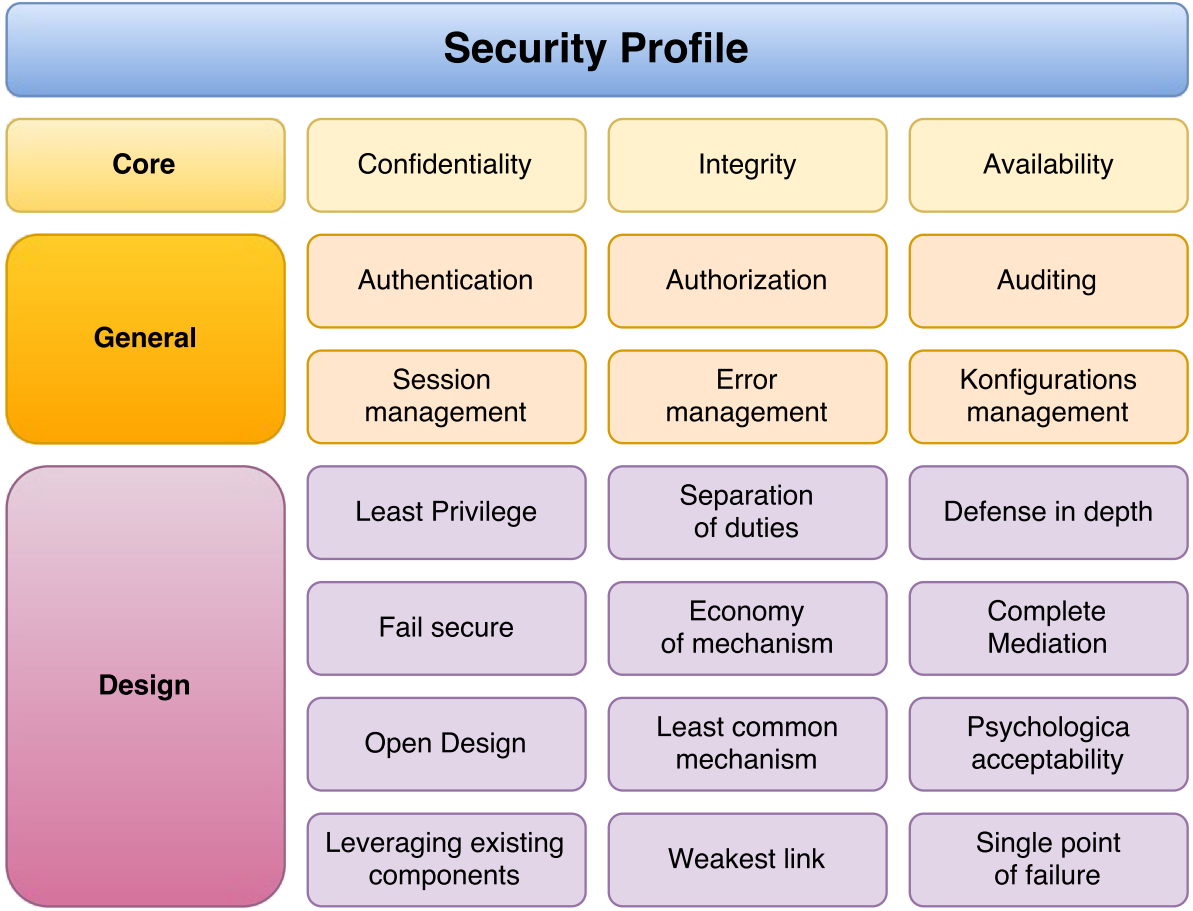


Goals of Protection

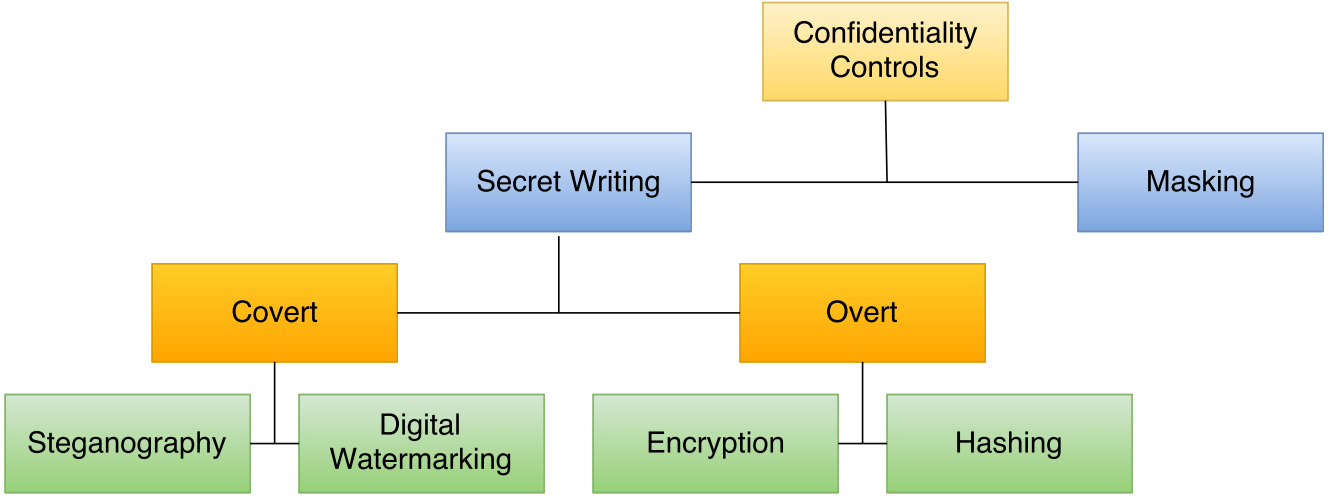
1. Confidentiality
2. Integrity
3. Availability
4. Non-repudiation
5. Authenticity
6. Privacy

secure systems achieve all of these goals of protection

Security Profile of Software



Confidentiality Requirements



- **In transit:** When the data are transmitted over unprotected networks
- **In processing**: When the data are held in computer memory or media for processing
- **In storage:** When the data are at rest, within transactional systems as well as non-transactional systems including archives

Confidentiality Requirements

Examples:

- "Personal health information must be protected against disclosure using approved encryption mechanisms."
- "Password and other sensitive input fields need to be masked."
- "Passwords must not be stored in the clear in backend systems and when stored must be hashed with at least an equivalent to the SHA-256 hash function."
- "Transport layer security (TLS) such as Secure Socket Layer must be in place to protect against insider man-in-the-middle (MITM) threats for all credit card information that is transmitted."
- "The use of nonsecure transport protocols such as File Transfer Protocol (FTP) to transmit account credentials in the clear to third parties outside your organization should not be allowed."
- "Log files must not store any sensitive information as defined by the business in humanly readable or easily decipherable form."

Integrity Requirements

Two basic types:

- System integrity
- Data integrity

Integrity Tools:

- Input validation
- Parity bit checking
- cyclic redundancy check (CRC)
- (cryptographic) Hashing

Examples:

- All input forms and query string inputs need to be validated against a set of allowable inputs before the software accepts it for processing."
- "Software that is published should provide the recipient with a computed checksum and the hash function used to compute the checksum, so that the recipient can validate its accuracy and completeness."
- "All nonhuman actors such as system and batch processes need to be identified, monitored, and prevented from altering data as it passes on systems that they run on, unless explicitly authorized to."

Availability Requirements

High-Availability Requirements as Measures of Nines

Measurement	Availability (%)	Downtime per year	Downtime per month (30 days)	Downtime per week
Three nines	99.9	8.76 hours	43.2 min	10.1 min
Four nines	99.99	52.6 min	4.32 min	1.01 min
Five nines	99.999	5.26 min	25.9 s	6.05 s
Six nines	99.9999	31.5 s	2.59 s	0.605 s

Availability Requirements

Examples:

- "The software shall ensure high availability of five nines (99.999%) as defined in the SLA."
- "The number of users at any one given point of time who should be able to use the software can be up to 300 users."
- "Software and data should be replicated across data centers to provide load balancing and redundancy."
- "Mission critical functionality in the software should be restored to normal operations within 1 hour of disruption; mission essential functionality in the software should be restored to normal operations within 4 hours of disruption; and mission support functionality in software should be restored to normal operations within 24 hours of disruption."

Authentication Requirements

Basic generic requirement:

- ❑ Validate an entities claim

This means:

- ❑ verify and assure the legitimacy and validity of the identity that is presenting entity claims for verification

Howto? Authentication!

Types:

- ❑ Anonymous
- ❑ Basic
- ❑ Digest
- ❑ Integrated
- ❑ Client certificates
- ❑ Forms
- ❑ Token
- ❑ Smart cards
- ❑ Biometrics

Authorization Requirements

Basic generic requirement:

- ❑ confirm that an authenticated entity has the needed rights and privileges to access and perform actions on a requested resource

Howto? Access control!

Types:

- ❑ Discretionary access control (DAC)
- ❑ Nondiscretionary access control (NDAC)
- ❑ Mandatory access control (MAC)
- ❑ Role based Access Control (RBAC)
- ❑ Resource Based Access Control

Typical rights:

- ❑ CRUD (create, read, update, delete)

Auditing/Logging Requirements

Bare minimum:

- ❑ Identity of the subject (user or process) performing an action (who)
- ❑ Action (what)
- ❑ Object on which the action was performed (where)
- ❑ Timestamp of the action (when)

Examples:

- ❑ "All failed logon attempts will be logged along with the timestamp and the Internet Protocol address where the request originated."
- ❑ "A before and an after snapshot of the pricing data that changed when a user updates the pricing of a product must be tracked with the following auditable fields—identity, action, object and timestamp."
- ❑ "Audit logs should always append and never be overwritten."
- ❑ "The audit logs must be securely retained for a period of 3 years."

Session Management Requirements

Sessions are an integral part of Role-Based-Access-Management (RBAC) and necessary for a number of features based on user interaction

Challenge:

- ❑ Provide session management on stateless infrastructure

Examples:

- ❑ "Each user activity will need to be uniquely tracked."
- ❑ "The user should not be required to provide user credential once authenticated within the Internet banking application."
- ❑ "Sessions must be explicitly abandoned when the user logs off or closes the browser window."
- ❑ "Session identifiers used to identify user sessions must not be passed in clear text or be easily guessable."

Error and Exception Management Requirements

Errors and exceptions are potential sources of information disclosure. Verbose error messages and unhandled exception reports can result in divulging internal application architecture, design, and configuration information. Sessions are an integral part of Role-Based-Access-Management (RBAC) and necessary for a number of features based on user interaction

Examples:

- "All exceptions are to be explicitly handled using try, catch, and finally blocks."
- "Error messages that are displayed to the end user will reveal only the needed information without disclosing any internal system error details."
- "Security exception details are to be audited and monitored periodically."

Error and Exception Management Requirements

Errors and exceptions are potential sources of information disclosure. Verbose error messages and unhandled exception reports can result in divulging internal application architecture, design, and configuration information. Sessions are an integral part of Role-Based-Access-Management (RBAC) and necessary for a number of features based on user interaction

Examples:

- "All exceptions are to be explicitly handled using try, catch, and finally blocks."
- "Error messages that are displayed to the end user will reveal only the needed information without disclosing any internal system error details."
- "Security exception details are to be audited and monitored periodically."

Configuration Parameters Management Requirements

Manipulation of software configuration parameters is a common attack vector and potential starting point for further malicious acitivities

Examples:

- The Web application configuration file must encrypt sensitive database connections settings and other sensitive application settings."
- "Passwords must not be hard coded in line code."
- "Initialization and disposal of global variables need to be carefully and explicitly monitored."
- "Application and/or session OnStart and OnEnd events must include protection of configuration information as a safeguard against disclosure threats."

Sequencing and Timing Requirements

Sequencing and timing design flaws in software can lead to what is commonly known as race conditions or time of check/time of use (TOC/TOU) attacks.

Prerequisite for race conditions

- ❑ Concurrency
- ❑ Shared objects
- ❑ State Changes

What might happen:

- ❑ Undesirable sequence of events, where one event that is to follow in the program execution order attempts to supersede its preceding event in its operations.
- ❑ Multiple unsynchronized threads executing simultaneously for a process that needs to be completed atomically.
- ❑ Infinite loops that prevent a program from returning control to the normal flow of logic.

How to protect

- ❑ Avoid race windows
- ❑ Atomic operations
- ❑ Mutual exclusion (Mutex)

Archiving Requirements

Archiving requirements might be there for reasons of business continuity or as a regulatory requirement

Examples

- Where will the data or information be stored?
- Will it be in a transactional system that is remote and online or will it be in offline storage media?
- How much space do we need in the archival system?
- How do we ensure that the media is not rewritable? For example, it is better to store archives in read-only media instead of read–write media.
- How long will we need to store the archives for?
- Is there a regulatory requirement to store the data for a set period of time?
- Is our archival retention policy contradictory to any compliance or regulatory requirements?
- In what format will the data or information be stored? Clear text or cipher text?
- If the data or information are stored in cipher text, how is this accomplished and are there management processes in place that will ensure proper retrieval?
- How will these archives themselves be protected?

Deployment Environment Requirements

Security Requirements might change substantially based on the Deployment Environment

Examples

- ❑ Will the software be deployed in an Internet, Extranet, or Intranet environment?
- ❑ Will the software be hosted in a demilitarized zone (DMZ)?
- ❑ What ports and protocols are available for use?
- ❑ What privileges will be allowed in the production environment?
- ❑ Will the software be transmitting sensitive or confidential information?
- ❑ Will the software be load balanced and how is clustering architected?
- ❑ Will the software be deployed in a Web farm environment?
- ❑ Will the software need to support SSO authentication?
- ❑ Can we leverage existing operating system event logging for auditing purposes?

What-else-requirements?

Further requirements:

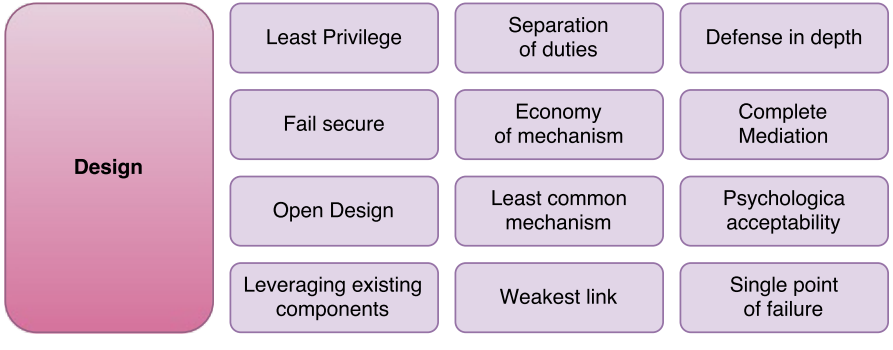
- ❑ International Requirements
- ❑ Procurement Requirements
- ❑ Antipiracy Requirements

Security Design Principles

US-CERT-Definition

Least Privilege

Only the minimum necessary rights should be assigned to a subject that requests access to a resource and should be in effect for the shortest duration necessary (remember to relinquish privileges). Granting permissions to a user beyond the scope of the necessary rights of an action can allow that user to obtain or change information in unwanted ways. Therefore, careful delegation of access rights can limit attackers from damaging a system.

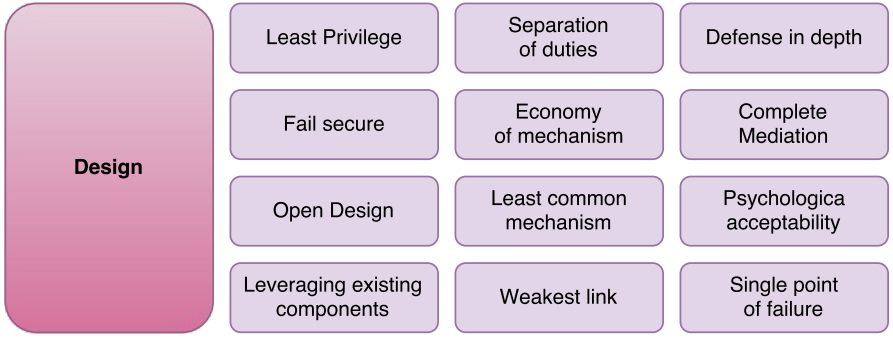


Security Design Principles

Designprinzipien (nach US-CERT-Definition)

Separation of duties

Separation of duty, as a security principle, has as its primary objective the prevention of fraud and errors. This objective is achieved by disseminating the tasks and associated privileges for a specific business process among multiple users. This principle is demonstrated in the traditional example of separation of duty found in the requirement of two signatures on a cheque.

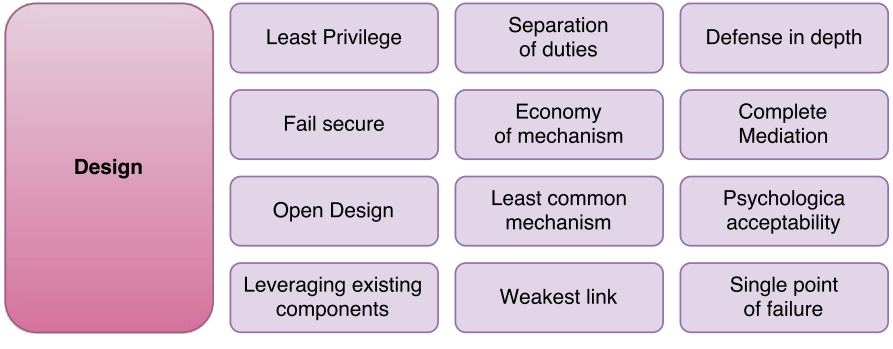


Security Design Principles

Designprinzipien (nach US-CERT-Definition)

Defense in depth

Layering security defenses in an application can reduce the chance of a successful attack. Incorporating redundant security mechanisms requires an attacker to circumvent each mechanism to gain access to a digital asset. For example, a software system with authentication checks may prevent an attacker that has subverted a firewall. Defending an application with multiple layers can prevent a single point of failure that compromises the security of the application.

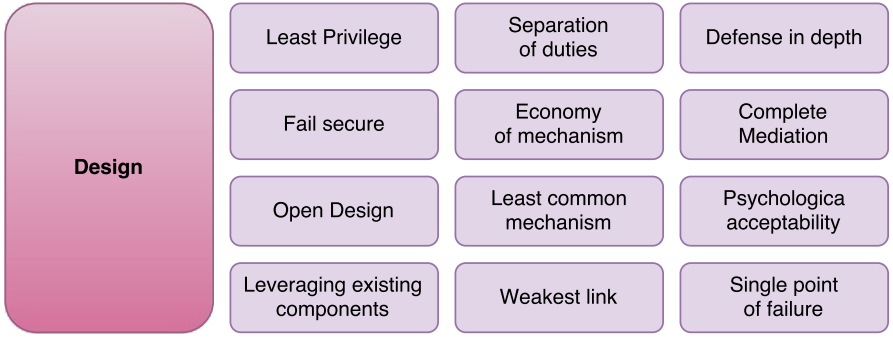


Security Design Principles

Designprinzipien (nach US-CERT-Definition)

- **Fail Secure**

  When a system fails, it should do so securely. This typically involves several things: secure defaults (default is to deny access); on failure undo changes and restore to a secure state; always check return values for failure; and in conditional code/filters make sure that there is a default case that does the right thing. The confidentiality and integrity of a system should remain even though availability has been lost.

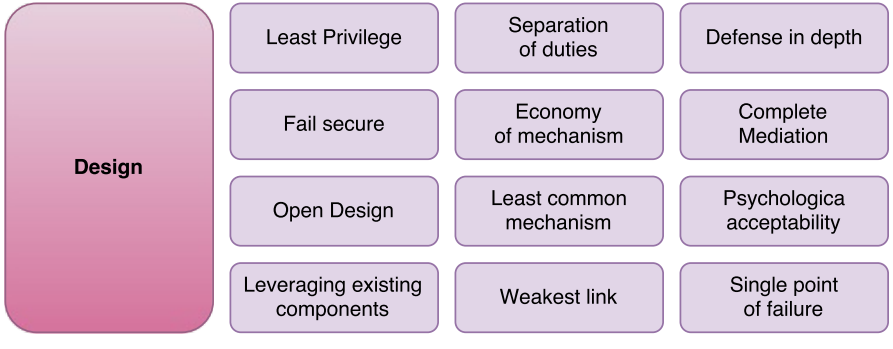


Security Design Principles

Designprinzipien (nach US-CERT-Definition)

❑Economy of mechanism

One factor in evaluating a system's security is its complexity. If the design, implementation, or security mechanisms are highly complex, then the likelihood of security vulnerabilities increases. Subtle problems in complex systems may be difficult to find, especially in copious amounts of code.

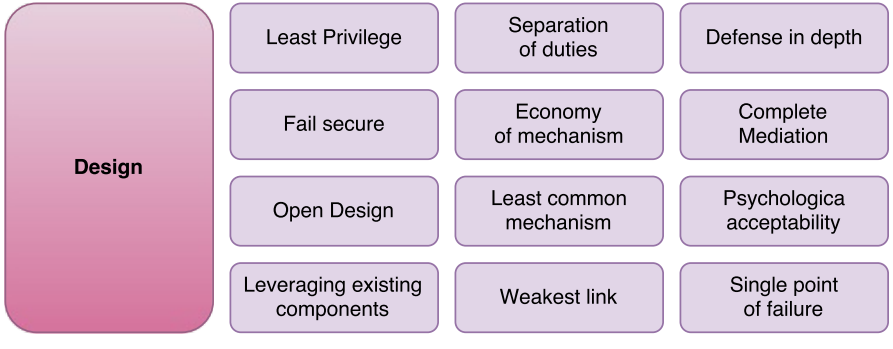


Security Design Principles

Designprinzipien (nach US-CERT-Definition)

❑Complete Mediation

A software system that requires access checks to an object each time a subject requests access, especially for security-critical objects, decreases the chances of mistakenly giving elevated permissions to that subject. A system that checks the subject's permissions to an object only once can invite attackers to exploit that system.

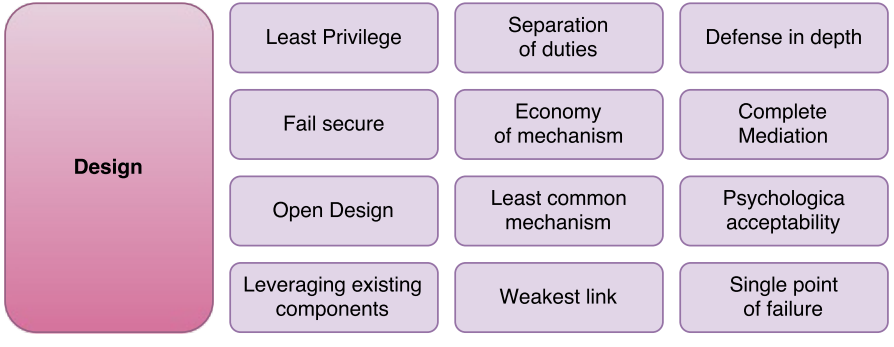


Security Design Principles

Designprinzipien (nach US-CERT-Definition)

Open Design

Open design: The design should not be secret. The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, and more easily protected, keys or passwords.

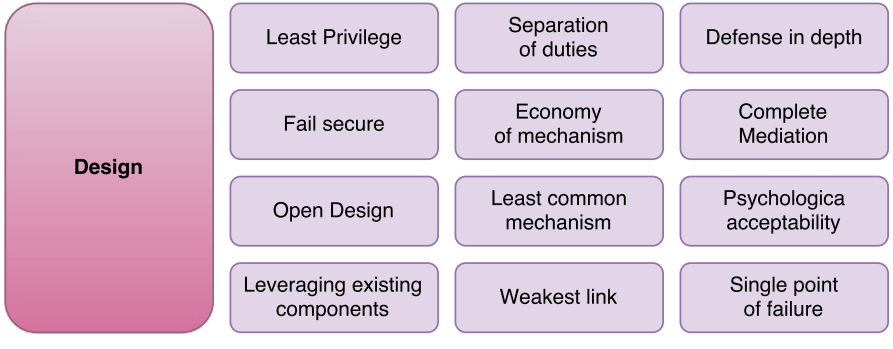


Security Design Principles

Designprinzipien (nach US-CERT-Definition)

- **Least common mechanism**

  Avoid having multiple subjects sharing mechanisms to grant access to a resource. For example, serving an application on the Internet allows both attackers and users to gain access to the application. Sensitive information can potentially be shared between the subjects via the mechanism.

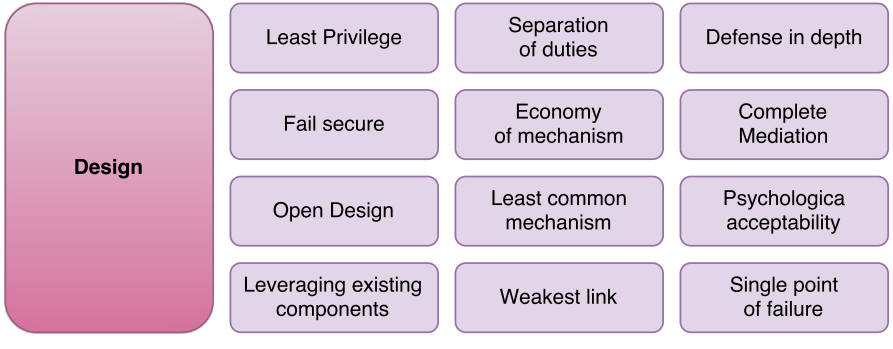


Security Design Principles

Designprinzipien (nach US-CERT-Definition)

- **Psychological acceptability**

  Accessibility to resources should not be inhibited by security mechanisms. If security mechanisms hinder the usability or accessibility of resources, then users may opt to turn off those mechanisms. Where possible, security mechanisms should be transparent to the users of the system or at most introduce minimal obstruction. Security mechanisms should be user friendly to facilitate their use and understanding in a software application.

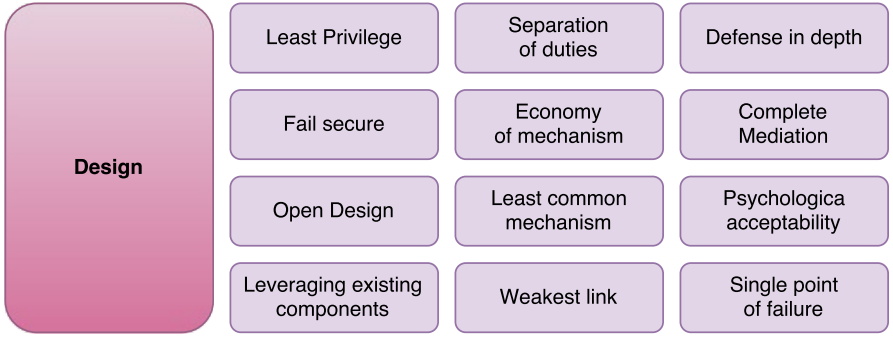


Security Design Principles

Designprinzipien (nach US-CERT-Definition)

❑Leveraging existing compoenents

This is a security principle that focuses on ensuring that the attack surface is not increased and no new vulnerabilities are introduced by promoting the reuse of existing software components, code and functionality.

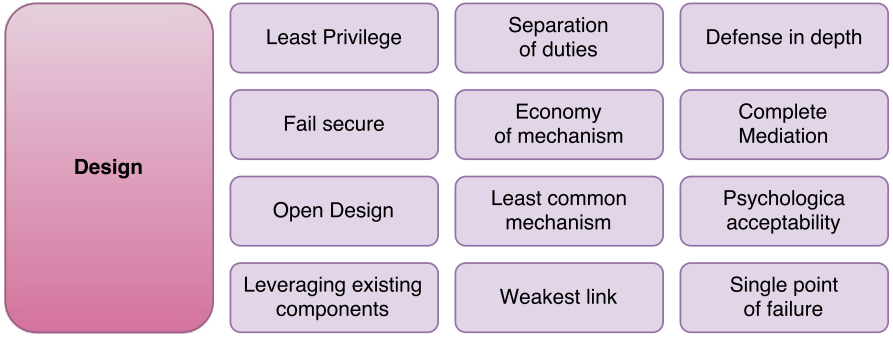


Security Design Principles

Designprinzipien (nach US-CERT-Definition)

❑Weakest link

Attackers are more likely to attack a weak spot in a software system than to penetrate a heavily fortified component. For example, some cryptographic algorithms can take many years to break, so attackers are not likely to attack encrypted information communicated in a network. Instead, the endpoints of communication (e.g., servers) may be much easier to attack. Knowing when the weak spots of a software application have been fortified can indicate to a software vendor whether the application is secure enough to be released.

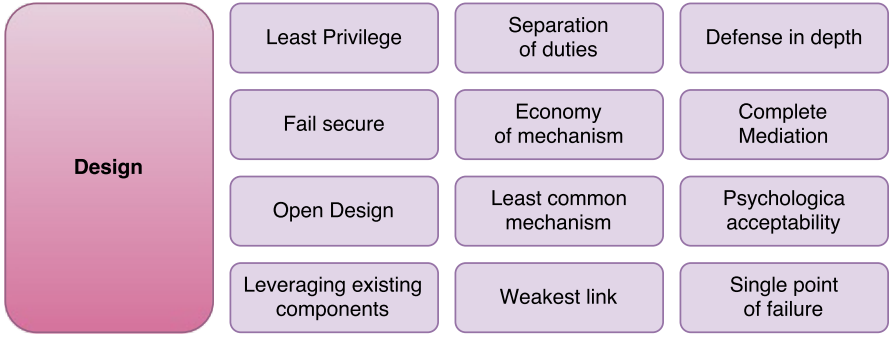


Security Design Principles

Designprinzipien (nach US-CERT-Definition)

Single point of failure

A single point of failure (SPOF) exists when a hardware or software component of a system can potentially make an application unavailable to users. Highly available systems tend to avoid a single point of failure by using redundancy in every operation.



Terms and Definitions

- **vulnerability:**

  A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy.

- **attack:**

  automated scans don't count

  An assault on system security that derives from an intelligent threat, i.e., an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system.

- **threat:**

  A potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm.

- **risc:**

  An expectation of loss expressed as the probability that a particular threat will exploit a particular vulnerability with a particular harmful result. (risc is measured in €)

Vulnerabilities

General types of vulnerabilities:

- conceptional weaknesses e.g.
  - no security classification of data
  - improper role-models
  - weak authentication
  - missing policies for USB-sticks, mobile phones etc…
- configuration errors e.g.
  - wrong boot order, no bios password
  - open ports on firewalls
  - weak encryption algorithms allowed
  - missing patchmanagement
- insecure programming (that's what the lessons are about!)
  - innumerable mistakes!

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG
INFORMATIK UND MATHEMATIK

Vulnerabilities

❑Top 25 Most Dangerous Software Errors http://cwe.mitre.org/top25/

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1▼

Vulnerabilities

❑Top 25 Most Dangerous Software Errors http://cwe.mitre.org/top25/

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1▲
16	CWE-862	Missing Authorization	5.53	1	+2▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3▲

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

INFORMATIK UND MATHEMATIK

Brainteaser

- Find the problem!

```
#include <iostream>
#include <stdlib.h>
#include <string.h>

using namespace std;


int main()
{
    char* a = (char*) malloc (20);
    char* b = (char*) malloc (20);

    strcpy(b,"Secure Coding");
    strcpy(a,"Insecure Coding");

    a=b;

    cout << a << endl;
    cout << b << endl << endl;

    free (a);
    free (b);


    return 0;
}
```

"Writing in C or C++ is like running a chain saw with all the safety guards removed," *Bob Gray.*

thanks for your interest

to be continued