Security Engineering
TUM Summer Term 2014
Lecturer: Alexander Pretschner

Janosch Maier

29. Juli 2014

Inhaltsverzeichnis

1 Introduction

1.1 What is security?

1.1.1 Essential Security Goals

- **C**onfidentiality
- **I**ntegrity
- **A**vailability

of data and systems.

1.1.2 More Security Properties

- Non-repudiation (Nicht-Abstreitbarkeit)
- Auditability
- Accountability
- Privacy
- Anonymity

1.1.3 Does it Matter?

- Private data
- Commercial data
- Government data

1.1.4 Relevance and Challanges

- Security-sensitive applications: eVoting, car2car, car2internet, ...
- Economic perspective: Enablers & Drivers
- Fight against vulnerabilities, potential damages, cyber crime
- National interest
- Privacy issues
- Lack of standards & understanding
- Social engineering
- System Vulnerabilities – Password Management, Operating Systems, Software Bugs, Unchecked User Input

1.1.5 Do we really want security

- Makes system hard / slow to use
- Costs
- Risks
- Is security subproblem of risk analysis?
- Different kinds of systems
- Tradeoff with liberty

1.1.6 What to do?

- Technically – Security Engineering, Cryptography
- Organizationally – Security policies
- People-Related
- Legally

1.2 What is security engineering?

- Software Engineering + Information Security
- Malice, Error, Mischance
- Tools, Processes & methots to design, implement, test & evolve systems

1.2.1 Security vs Safety

- Safety: Failures in normal operation
- Security: Failures as consequences of a hacker

2 Introduction to Information Security

2.1 Cryptography

- Cryptography: Mathematical techniques concerning data security
- Kerckhoffs' principle: Secure, if adversary knows everything except declared secrets (keys)

2.1.1 Terminology

- $\sum$: Alphabet
- M: Message space
- C: Ciphertext space
- K: Key space
- $E_e : M \to C; e \in K$: Encrpytion Function (bijective)
- $D_d : C \to M; d \in K$: Decryption Function (reverse of E)
- $D_d(E_e(M)) = M \Leftrightarrow D_d = E_e^{-1}$: Encrpytion sheme / cipher
- (e, d): Keypair (Symmetric if d can easily derived from e. Mostly: $e = d$)
- Breakable, if m can be obtained from c without (e, d) in appropriate time.

2.2 Symmetric Cryptographyc Protocols

2.2.1 One-Time Pad

- Key at least as long as message
- Unbreakable, unless: Key intercepted, Key not random, Keys reused
- Integrity not preserved

2.2.2 Monoalphabetic Substitution Ciphers

- Substitute plaintext letter with cyphertext letter
- Letter / Digram Frequency

2.2.3 Homophonic Substitution Ciphers

- Substitute one letter in m by with more letters in c
- Frequency analysis no longer easily possible

2.2.4 Polyalphabetic Substitution Ciphers

- Different cyphers depending on position within the block
- Problem: Determine block length, then frequency analysis

2.2.5 Simple Transposition Ciphers

- Permutation of the letters within a block

2.2.6 Composition of Ciphers, Product Ciphers

- Security depending on used original ciphers
- Example: DES and 3DES
- Security ideas for block ciphers:
  - Ciphertext bit relys on all plaintext bits
  - No statistical relationship between m and c
  - Altering any plaintext bit alters ciphertext bit with propability .5
  - Altering ciphertext bit alters plaintext in unpredictable way
- 3DES: $c = E_{e1}(D_{e2}(E_{e3}(m)))$
  - DES slow to implement in software
  - $E_{e1} \circ E_{e2}$ gives more possible ciphertexts as $E_e \forall e \in K$
  - Meet-In-the-Middle attack possible

2.2.7 Stream Ciphers

- Keystream generated with cyphertext
- Keystream can be reproduced when m and c is known

2.2.8 MAC

- One-Way-Function: Easy to get hash value, but hard to get plaintext
- Non-invertibility, no hints from output to input, freedom of collision
- $f : M \times K \to T$ (T = Tag = Hash-Value), Not possible to generate m' with same tag t as m

2.3 Asymmetric Cryptographyc Protocols

- No shared Key
- Key-Generator + Encryption Algorithm + Decryption Algorithm
- E.g. PGP, S/MIME, SSH, IPSec, ...

2.3.1 RSA

- Create primes: p, q
- $n = pq$, $\phi = (p-1)(q-1)$
- Chose $e, 1 < e < \phi$, such that $ged(e, \phi) = 1$
- Compute d, such that $ed \equiv 1 \mod \phi$
- Private key = d, Public key = (n, e)
- Encryption: $c = m^e \mod n$
- Decryption: $m = c^d \mod n$
- Computation of exponents is cheap, Factoring of large numbes is hard
- Attack people / side-channels, not algorithms

2.3.2 Signatures

- Encrypt hash with private key
- Anybody can encrypt with public key and compare hash values

2.4 Key Management

- Symmetric: Agree on a key
- Asymmetric: Make sure, that a public key belongs to somebody

2.4.1 Trusted Authority

- Shared key between TA with everybody
- All messages pass TA: TA as Bottleneck (always online), Who can be trusted?
- TA used, to establish secure channel: A asks TA for session key. Session key encrypted for A and B send to A, A forwards session key
- TA used, to establish secure channel: A asks TA for session key. Session key encrypted for A send to A, session key encrypted for B send to B
- Problem: Replay attacks $\rightarrow$ Nonces

2.4.2 Needham-Shroeder Protocol

- $A \rightarrow TA$: $\{A, B, n_1\}$
- $TA \rightarrow A$: $\{A, B, n_1, k_{session}, \{A, k_{session}\}_{k_B}\}_{k_A}$
- $A \rightarrow B$: $\{A, k_{session}\}_{k_B}$
- $B \rightarrow A$: $\{n_2\}_{k_{session}}$
- $A \rightarrow B$: $\{n_2 - 1\}_{k_{session}}$

- Problem: Man in the Middle attack, if Eve gets Session Key
  - $E \to B$: $\{A, k_{session}\}_{k_B}$ [copied]
  - $B \to A$: $\{n_3\}_{k_{sesion}}$ [intercepted by Eve]
  - $E \to B$: $\{n_3 - 1\}_{k_{session}}$
- Solution: Timestamp in the message form TA to A that shall be forwarded to B. But requires synchronized clocks (e.g. Kerberos with Authorisation Server and Ticket Granting Server)
- Solution: Otway-Reas – Each number is attached to specific protocol run

2.4.3 Diffie-Hellman Key Agreement

- Prime p and Generator g publicly known, x_A, x_B secretly choosen
- $y_A = g^{x_A}(\mod p) \mid y_B = g^{x_B}(\mod p)$
- $k_{AB} = y_B^{x_A}(\mod p) \mid k_{AB} = y_A^{x_B}(\mod p)$
- One-Way function (discrete logarithm problem)
- But: Identity not known

2.4.4 Certificates

- $A \to B$: $A, \{\{k_{session}\}d_A\}e_B$
- Only works, if public key is known
- Man-in-the-middle attack when public key is retrieved from server, No binding between key and person
- PKI: Obtain authenticated public keys. Certificate is digitally signed statemnt that binds public key to an entity.
- $Cert_{C,A}$: C certifies the authenticity of A. B must trust C.
- Less keys needed. Problems: Revocation, Identity verification, Establishment of trust
- Semantics of Certificates
  1. A claims towards TA, that P_A is her public key
  2. $1 + T$ confirms it verified that A knows the corresponding private key
  3. Non-repudiation: A is liable for signed statements with the certified key
- Certificates only states something about public key, not about trustworthiness of an entity
- Solve recursive problem with Certificate Chains

- PKI: Hierachial certification (X.509), Cross-certification, Unstructured certification (PGP, WoT)
- Purpose of PKIs: Generation of keypairs, Authentication of entities, Generation/Distribution/Revocation/Verification of certificates
- Public Keys for encryption, authentication, non-repudiation
- Problems: Organisation, Chicken-and-egg, Standardization, Legal Problems, Business model, Naming problem, Expiration & Revocation

2.5 Applications: Digital Signature & Encrypted Mails

- PGP: secure email exchange, trust in keys not people, web of trust, revocation possible
- X.509: Bind certificates to real names, CRLs, Structured certificates
- S/MIME: Secure Multipurpose Internet Mail Extensions, Relies on X.509 (or PGP – not compatible), Multipart/Signed, Multipart/encrypted
- TLS/SSL: Use of Certificates, Needham-Schroeder

2.6 Access Control

2.6.1 Access Control Matrix Model

- Subjects S want to access objects O. Generally $S \subseteq O$
- Abstract rights R (e.g. read, write, own, execute)
- Matrix A: $S \times O \rightarrow 2^R$
- E.g. unix filesystem
- Store as columns: ACL, Store as lines: Capabilities List

2.6.2 RBAC

- ACLs unmanagable $\rightarrow$ Roles (Set of permissions)
- See summary "Computergestützte Gruppenarbeit". *Very exam relevant*

2.6.3 Formalisms

- Programs/systems describe state (memory, registers, ...) transitions. Protection states: Part of the states concerning permissions.
- Context-awareness: Breaking-glass policy (Study shows, that 85% gets emergencies) / Circumvention
- Protection state transitions: Create/Destroy subject, Create/Destroy object, Enter/Delete Right
- Leakage / Undecidability: If a right r is added to an element, this right is leaked. If a system can never leak the right r, it is safe w.r.t. r. $\Rightarrow$ Whether a given state is save is an undecidable problem

- Security Policies: Partition states into authorized/secure, unauthorized/unsecure

2.6.4 MAC / DAC

- Discretionary Access Control: Somebody can change the rights of access: Unix file system, User own resources and control their access, Access based on subjects and objects – flexible but open to mistakes/abuse
- Mandatory Access Control: System entity controls access to data, Policies not managed by users
  - Read Down: Subject clearance $\geq$ object clearance
  - Write Up: Subject clearance $\leq$ object clearance
  - Integrity: no read-up and write-down
  - Lattice: Hierachial (linear) ordering & set of categories $\Rightarrow$ Lettuce is partial order on labels: $c(h1, c1) \leq (h2, c2)$ iff $h1 \leq h2$ and $c1 \subseteq c2$.
  - Confidentiality (Information Flow) Policies

2.6.5 Bell-LaPadula model

- MAC, subjects/objects classified by security levels, labels do not change: multilevel security, no write-down, no read-up
- No-write-down prevents high-level subjects leaking messages
- Advantages: Modeling confidentiality in operating systems / databases
- Problems: Static model, Not specified, how add / delete objects, Contains covert channels (e.g. Sidechannel attack)

2.6.6 Biba model

- Integrity problem. Based on no-read-down, no write-up
- Relax no-read-down (subject low watermark)
- Relax no-write-up (object low watermark)
- Change read/write directions

2.6.7 Chinese-Wall Policies

- Conflicts because clients of one company are direct competitors in same market

$\Rightarrow$ No information flow that causes conflict of interest

- Companies C, subjects S, objects O
- $cd : O \to C$: company dataset of ebjects
- $cic : O \to 2^C$

- Security label: $(cic(o), cd(o))$
- Security matrix $N = S \times O$, $N(s, o)$ is true, if s had access to o, Initial state: everything is false, Access permissions change dynamically
- ss-propperty: s is pertmitted access to o iff $\forall o'$ with $N(s, o')$ trie it holds $cd(o) = cd(o')$ or $cd(o) \notin cic(o')$
- Problem: Indirect information flow (internal file-server, ...)

2.7 Usage Control

What happens after distributions of data (rights and duties): e.g. Personal data, intellectual property, business data, administrative secrets

2.7.1 Roles and Classes of Requirements

- Data provider & data consumer (consumer may also be provider)
- Provider defince policies (typically without access over consumer)

2.7.2 Control and Observation

- Preventiv / Control: Try to enforce control (DRM)
- Detective / Observation: Detect violation (Law)

2.7.3 Requirements

- Restrictions and neccessary actions (Permission and duty)
- Conditions (Time, purpose, ...)

2.8 Information Flow

- Information flow (data leakage) occurs, when value of a secret variable influences another (non secret) value
- Security policies describe which flows are allowed
- Formal Model of Non-Interference: Information flow from p1 to p2 if some actions of p1 influence p2. Problem: No interaction depending on secret data. (Not covered channels e.g. timing)

2.8.1 Non-Interference

- M = (S, A, O, step, output, s_0) – States S, actions A, outputs O, initial state s_0, function step: $S \times A \to S$, Output function $S \times A \to O$; run $S \times A^* \to S$ (Sequences of actions)
- System M + security domains D = { u, v }. Noninterference relation antireflexive on $D \times D$: Value of output(b) independent from executing a or not.

- Purge: $A^* \times D \rightarrow A^*$: Removes all actions from sequence, that shall not interfere with specified domain
- M is secure, if test(α, a) = test(purge(α, dom(a)), a) – System secure, iff output of action is independent from actions that shall not interfere with dom(a)
- Transitive policies (= multilevel security policy) not expressable in this model

2.8.2 Information Flow for Programs

- All runs: static analysis
- One run: dynamic analysis
- Problem: Label creeps (Everything depends on everything)

2.8.3 Information Flow Detection

- Explicit Information flow from x to y: $y := f(x_1, ..., x_n)$
- Implicit Information flow from secret to x: if (secret) x := 1; else x := 2
- Implicit Information flow from secret to x and y: if (secret) x := 1; else y := 1 (regardless of branch)

2.8.4 Abstraction Levels

- Systems
- Programs (System-Calls?)
- Examples: SELinux

3 Software Engineering meets Security

3.1 Security Requirements

- Properties of a system
- Non-functional requirements: Security, Usability, ...
- Quantified security indicators? ⇒ Need to be made precice
- Threat models: Protection agains threats you have thought about

3.2 Requirements Engineering

- Activities: Elicitation, Analysis, Specification, Validation ⇔ Design, Implementation, V&V

3.3 Use cases & Misuse cases

- Sequence of steps between user and system
- Set of scenarios for common user goal
- Generally: Functional requirements, High level, Discussions ⇒ How about Security Requirements
- Misuse Cases: Wanted vs. unwanted behavior (sometimes unintentionally)
- Use & Misusecases can be structured in one diagram

3.4 Refinement

3.4.1 Fault Tree Analysis

- Top-Down: What needs to happen for bad things to appear?
- Bottom-Up: What happens, if one component produces unintended results / fails

3.4.2 Attack Trees

- Fault-Trees for Attacks. Nodes as threats
- What is needed to perform an attack?

3.4.3 Discussion

- Security requirements mostly for complete system, sometimes components
- Probability for security breaches not predictable
- Reliability Engineering: Estimate propability of failures
- Scenario Analysis, Risk analysis

3.5 Regulations as Requirements

3.5.1 Bundesdatenschutzgesetz

- Protection of individuals from being constrained in their personal rights by use of personal data
- Personal data: Identitiy can be seduced, Partial sensitive: racial, ethnical, ... status ⇒ Explicit consent necessary
- 7 Principles of data protection
  - Appropriation (Zweckbindung)
  - Minimum possible extent (Datensparsamkeit)
  - Data Security (Access, Usage, ...)
  - Data Secrecy (Non-authorized people not allowed to collect/process/use)
  - Responsibility (Who is responsible)
  - Individual decision (Einzelentscheidung)
  - Transparency (Auskunftsrecht)
- Collection, Handling, Usage of personal data is forbidden unless explicitly allowed (by law, by data owner)
- Information self-determination
- Private Institutions
  - Data protection officer, if 10 people (electronically or 20 manually) are steadily handling personal data
  - Collection, storage, ... for business purpose allowed (Justified by purpose / rightful interest / data publicly available)
  - Purpose of use specified
- Public Institutions
  - Allowed if necesarry to perform duties
  - If third party data, personal data optional
  - Strict regulations on further collection / transfer
  - Data protection officer compulsory

3.5.2 Individual Rights

- Cost-free access to stored data, origin, receivers, purpose of storage
- Data collected without knowledge ⇒ Notification
- Must be corrected if wrong
- Must be deleted (locked up in special circumstances) if purpose has served

3.6 Design-level security

Methodology to build secure applications: formal, general, usable, wide spectrum, tool supported, scales

3.6.1 Model Driven Security

- Model Driven Architecture: System Model (e.g. in UML), Model Transformation ⇒ Target System (Usable Code)
- Model Driven Security: System Model + Security Model, Model Transformation + Extensions ⇒ Target System + Security Infrastructure
- Model: View of the System (with semantics)
- MDA (Object Management Group Standard) based on Modeling (UML) / Metamodeling (Meta-Object Facitlity) standards
- UML: abstract/concrete syntax, includes Object Constraint Language, But not yet Formal Method, (e.g. Class Diagram, Statechart)
- Domain Specific Languages: Metamodel defines UML Model for a specific domain
- MDA Translation: Fix platform (e.g. J2EE/EJB), Translation produces JavaCode & XML deployment descriptors

3.6.2 Secure components

- Security Design Language (Abstract + Concrete syntax): e.g. RBAC + class diagrams
- Dialect bridges Design Language + Security Language (identify protected resources)
- Access Control Policies enforced using a reference monitor. Checks whether user are authorized to performs actions
- Access Control
  - Declarative: User u has Permission $p \Leftrightarrow (u, p) \in AC$ (User in role customer may withdraw money)
  - Programmatic: Assertion at relevant program points. System environment provides information needed for decisicon (if he is owner of the account)
  - RBAC (+ Extensions: Role/User/Permission Hierarchie, Authorization Constraints)
- SecureUML: Formalize Permissions for actions on protected resources
  - Roles & Users not design-time issue, but administrative
  - Permissions after language combination (actions & resources needed) bind action(s) to single resource (Association class between role and class)

  - Authorization Constraints in OCL subset (self, caller, attributes, side-effect free methods, ...) e.g. caller = self.owner.name

- ComponentUML: Class-based language for data modeling
- Combine SecureUML & ComponentUML: Dialect combines syntax, identify protected resources, resource action, define action hierarchie
  - Resources identified by subtyping
  - Resource actions defined using named dependencies from resource types to action classes

3.6.3 Semantics

- Intuitively: System behaves as before except certain actions are disallowed by access control
- Formally: Defined as labeled transition system (LTS, approx. state machine) ⇒ Constrains decrease set of possible traces

3.6.4 Generating security Infrastructures

- Decrease Burden, Faster adaption, Better Scaling, Correctness
- EJB
  - Deployment descriptor record with AC information
  - RBAC: Action → Method → Deployment-descriptor with permissions & security-roles
  - Assertion: Compute permissions & roles. Check if constraint attached & include assertion in method
- .NET
  - Language independency (communication of languages)
  - AC configured as attributes of methods

3.6.5 Secure controllers

- Controller defines system behaviour (states & events)
- 3-tier architecture: MVC.
- Metamodel: Statemachine formalizes Controller
- Generate web applications based on Java servlets

3.7 Security Patterns for Software & Systems

3.7.1 Security principles

- Least privilege: Minimize negative consequences of errors / attacks (e.g. keys in buildings)
- Complete mediation: Always control access (e.g. airports)
- Secure, fail-safe defaults: Start in secure state and return to secure state when failuring (e.g. whitelisting)
- Compartmentalization: Organization into isolated zones/compartments (e.g. virtual machine)
- Minimizing exposure: Reduce external interfaces (e.g. disable bluetooth)

3.7.2 Kind of patterns

- Common practice / blueprint / guide for design
- Requirements, Threats, Design, Business Processes, IT Infrastructure, IT Processes
- Requirements: Protection profiles e.g. cryptographic key management
- Threat modeling: Attack modeling means systematically documenting attacks (structured, reusable)
- IT controll: Auditing requirements

3.7.3 Security patterns

- Idea: Design patterns for secure system design
- Problem: security cross-cutting $\Rightarrow$ security pattern not within one part of software
- Groups of patterns: Security/Risk management, identification/authentication, (system / operating system) access control, accounting, firewall, secure internet applications
- Pattern Library: Name, Problem, Solution, Consequences
- Available (access to resources) System Catalog: Checkpointed System (Rollback), Replicated System
- Protected (protecting resources) System Catalog: Protected System (Reference Monitor), Policy, Secure Communication
- Secure Communication
  - Motivation: Ensure mutual security policy: Prevent eavsdropping, Modification of data, ...
  - Applicability: Communication parties, Communication channel, (Communication Protection Proxy)

  - Variants: In-line proxy (SSL), Proxy as out-of-band service (PGP)
  - Consequences: Data exchange protected, may reduce throughput/latency, may require cryptography (deployment), may interfere with other resources (routers, ...)
  - Implementation: Proxys may apply: Data Origin authentication, Data integrity (replay detection, ordering)

- Protected System (Reference Monitor / Enclave)
  - Motivation: Enforce access security policy, Complete Mediation
  - Variants: Centralized guard (syscalls), guard distributed (1 per resource type, Java 2 Security Architecture)
  - Uses "Facade" Design Pattern
  - Participants: Client, Guard, Policy, Resources
  - Consequences: Isolates resources, Loosens copuling between policy/-resource implementation, Improves system assurability, Degrades performance
  - Implementation: Guard self-protection, Assurance (of correct guard functionality)

3.7.4 Integration into development process

- Choose design, Identify system components, communication channels, etc.
- Apply "Available System Sequence": Identify critical components & apply Error correction / Fault-Tolerant-System / Replication Pattern
- Apply "Protected System Sequence": Identify resources and actors, define protected systems (PS), Chose guard for PS, define Policy pattern (Policy Decision Point)
- Review / Refactor

3.8 Implementation-level Security

3.8.1 Buffer overflows

- Buffers are \0-terminated. Problem writing more into a buffer than space is allocated
- Stack grows from top to bottom
- Defense
  - Canary: Check if local variable canary is changed before leaving a function
  - Defensive programming: Avoid unsafe functions "strncpy" instead of "strcpy". Check array bounds, ...
  - Avoid C(++): Use type safe languages (but speed, ...)
  - Avoid Buffers on stack (but heap overflows possible)
  - Mark stack (or heap) non-executable (but still return-to-libc possible)

3.8.2 Format string vulnerabilities

- Problem: Mixing of control ("e.g. %s") & data structures
- Less & more parameters for printf
- Arguments are pushed to the stack and then printed. If there are less arguments, printf reads from the stack and might print function data
- Function call without format string: $printf(s)$ with $*s$ = "%08x" prints first stack entry
- Function call $printf(\%x\%x\%x\%x)$ prints some chibberish and then the local variable on stack
- $\%n$ stores the number of characters printed into memory. Function call $printf(\%x\%x\%x\%x\%n)$ stores the length of all printed characters into the stack (destroy integrity)
- Read/Write arbitrary memory locations
- Solutions: Be aware of library functions, Analysis tools, Weaker library functions ⇒ Sanitizing functions

3.8.3 Data Injection

- Simple PHP data injection: Show files from filesystem
- SQL-injection: Usually in forms: Input of form "SELECT * FROM addressinfo WHERE user = 'alex' OR 1=1;'" ⇒ Prepared Statements (Query object instead of string), Strict type checking

3.8.4 Cross site scripting / Cross site request forgery

- Non-persistent attacks / Reflective attacks (e.g. Forged link): Put javascript code within a link, Javascript will be reflected by the webserver and run on the client
- Persistent attacks (e.g. Malicious guestbook content): Put javascript in a guestbook, any other visitor will run the script ⇒ Server should filter for embedded scripts
- DOM based attacks / Local attacks (web application not involved): Put javascript code within a script that is run locally by modifying DOM elements. Malicious code never embedded in raw htlm ⇒ Disable Javascript / Don't click on malicious links
- Cross Site Request Forgery (XSRF): Exploirt web site's trust in user's browser. Click malicious link (img src), while being logged into secure system ⇒ Command authentication, Lifetime of session cookies, http referrer, ...

4 Risk and system analysis & Risk assesment

4.1 Motivation and Goals

- Enable mission accomplishment (by secure IT systems, well-informed decisions)
- Balance cost of security measures with mission achievement
- Maintain confidence, Protect sensitive data, Avoid fraud, Avoid liability, Laws, ...

4.2 Assets, Threats, Vulnerabilities

- Assets: Things of value for company (Information, Products, Buildings, Shares, Systems, People, Reputation, Trust, Potential political fallout) – Tangible (physical, logical) vs. intangible (reputation)
- Value of information: Producing costs, Price on market, Reproduction price, Benefits, Business advantage, Loss of confidence
- Potential cause of unwanted event that may result in harm to organization & assets (Harm: acces, damage; Event: exploit, attack; Accidental / Intentional)
- Vulnerability: Characteristic of asset which can be exploited by threat (Weaknesses, ...)
- Need to determine most important assets: Threads – Assets – Consequences
- Sources: Hacking (black/white-head), Insiders, Accidential deletion, Environmental damage, ...
- Direct impact: Destruction, Corruption, Theft/Loss, Disclosure, Inappropriate use, Interruption of services
- Identify threats: Attack trees, ...

4.3 Risk

- Possibility to suffer harm or loss
- Measure of failure to counter a threat
- Characteristics: Loss, Likelihood, Degree of control
- Probability that specific threat successfully exploits vulnerability causing loss
- RISK = $\sum$ IMPACT/VALUE $\times$ PROBABILITY (Annual Loss Exposure – Problem: In case of event damage is much higher, Additional costs rebuying stuff)
- Handling riks: Avoiding (by changing requirements / System characteristics), Transferring (buy insurance), Assuming (accepting & controlling)

4.4 Qualitativ & Quantitative risk analysis & management

- Procedure: Identify assets; Ascertain threats, riks, concerns; Prioritize; Corrective measures; Monitor effectiveness
- Quantitative: Numeric values for value, Probability, ... ⇒ Numbers good for comparison, but critical knowledge base, costly
- Qualitative: "What if" questions (possibly categories), ⇒ simpler, easy involvement, but more subjection no basis for cost analysis
- Security rated with regard to the time it takes to breakt them given a set of tools

4.5 BSI baseline protection

- "Security Patterns in action": Sharing of ideas in security design / Standard measures
- Risk analysis only in extreme cases, standard measures in normal cases
- Structure IT assets into modules (components/procedures/IT systems) & map company's infrastructure to modules
- Establish IT security concept: Get IT structure, Qualitative analysis of protection requirements, Map infrastructure to manual components, Risk analysis for assets with very high risk, Compare situation to safeguard & take action
- Combination of requirements
  - Maximum principle (Two systems on one server: Protection maximum of the two)
  - Dependency relationships (Protection of A need to be high, if B depents on A and has high protection need)
  - Cumulative effects (Many apps with low protection, but server crash does harm)
  - Distributive effect (Irrelevant parts of "high" application may crash)
- Data will be mapped to system. ⇒ Protect the systems instead of the data. Threats are linked to safeguards.

4.5.1 Domain Concepts

- General IT security aspects (e.g. empolyees leaving the company)
- Infrastructure security (e.g. physical security)
- IT systems (e.g. firewalls
- Networking aspects (e.g. routers)
- Applications (e.g. web servers)

5 Evaluation Criteria: The Common Criteria

- How do you assess, that something is secure?
- "How do you convince s.b. that your system is secure?"
- Orange-Book hierarchic: No protection (D), DAC (C), MAC (B), Verification design
- Common Criteria: Design Secure Environment + Evaluation Assurance Levels
- Certifying Products (s.t. by looking at the process)
- Evaluation techniques: Analysis of process, Check that process is applied, Verification of proofs, Penetration testing, Analysis of vulnerabilities, ...
- Target of Evaluation (ToE, e.g. os, network, ...)
- Idea: standardized functional security requirements & assurance requirements $\Rightarrow$ Protection Profiles (consistent & complete)
- Vendor instanciates PP into security target (ST)
- Structure of CC: Introdcution + General model, Security functional requirements, Security assurance requirements
  - General model – protection profile: Security objectives instanciated by Security target. ToE shall confirm with ST. (PP $\rightarrow$ ST $\rightarrow$ Evaluate ToE)
  - Security functional requirements: Desired security behaviour for ToE (e.g. Security audit, communication, crypto support, user data protection, identification & authentication, security management, privacy, protection of ToE security functions)
  - ToE assurance requirements: How can security requirements be implemented (Configuration management, delivery and operation, development, guidance documents)
- Evaluation Assurance Levels: Functionally testest (EAL1) to Formally verified design (EAL7)
- Pros: Flexible & thorough, international, protection profiles as overview for implementation, EALs overview for assurance
- Cons: Evaluation relates to system version, full control over product is needed, timing (CC evaluation takes 1 year), terminology, cost-effectiveness?, quality of PPs?, Evaluation of documents instead of system

5.1 Microsoft SDL

- Microsoft Security Development Lifecycle
- Training, Requirements, Design, Implementation, Verification, Release, Response

- SDL-Agile: Sprint/One-Time Requirements
- In general: Think about security in each software development step