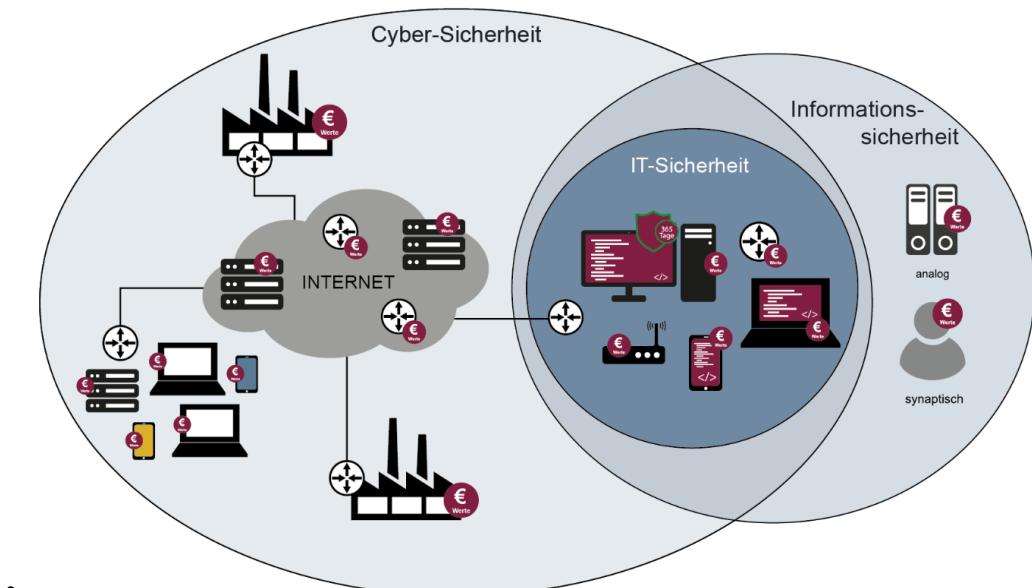


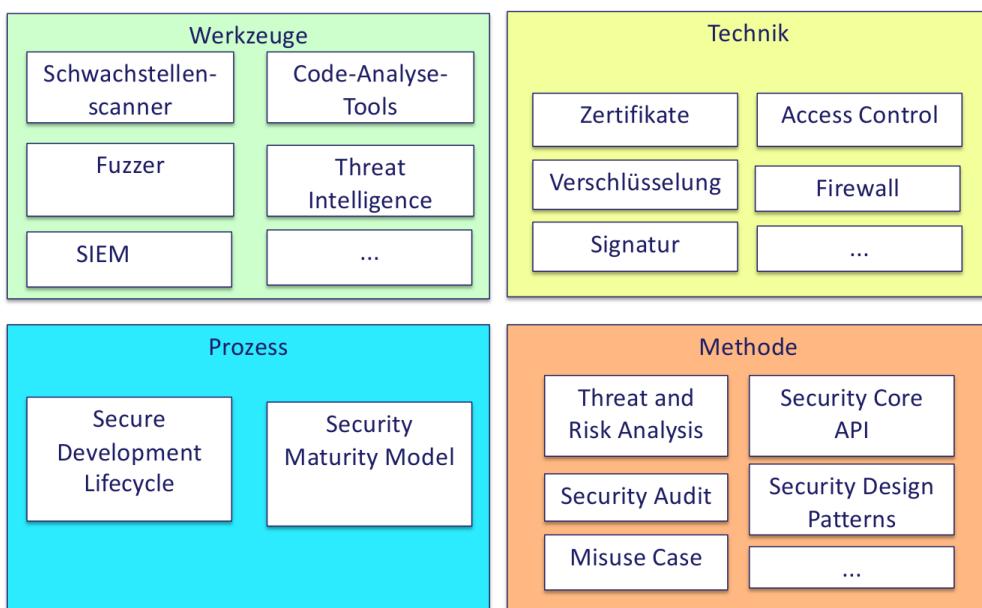
Zusammenfassung: Security-Engineering in der IT (SEIT)

1. Motivation und Definitionen

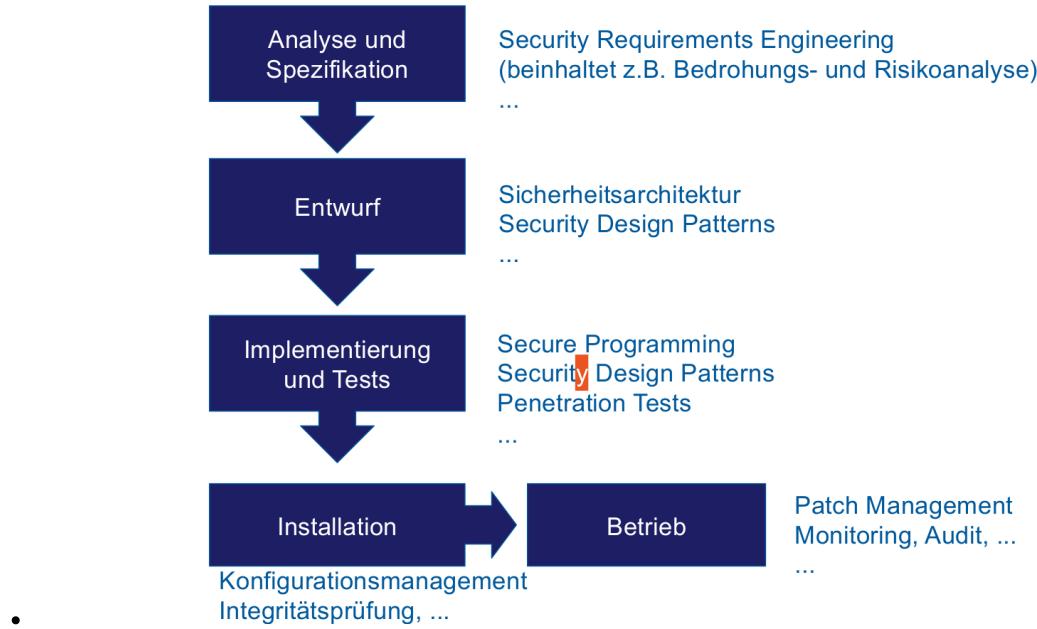
- Cyber Crime: Straftaten die moderne Informationstechnik und elektronische Infrastrukturen (aus-)nutze
 - Cyber Warfare: Grenzüberschreitende Cyber-Kriminalität, an der mindestens ein staatlicher Akteur beteiligt ist
 - Cyber Terrorism: Störrung / Behinderung v. Regierungen / Institutionen um bestimmte Ziele zu erreichen (z.B. Politik, Ideologie)
 - Cyber Espionage: Ziel geheime oder vertrauliche Daten zu erlangen
 - Hacktivism: Ziviles Ungehorsam um bestimmte Motivationen zu bewerben (z.B. Politk)
-
- Angriffsvektor: Methode / Technik um in System einzudringen / kompromittieren
-
- White Hat Hacker: Konstruktiver Hacker (Legal)
 - Black Hat Hacker: Destruktiv und meist mit krimineller Motivation
-
- IT-Sicherheit: Bereich der Informatik, Schutz von Systemen und Informationen, Umfasst Abwehr von mutwilligen und bösartigen Angriffen auf diese Schutzziele
 - System: Ausschnitt aus realen / gedanklichen Welt aus Gegenständen und vorhanden Strukturen, Nicht zerlegbare Systemteile sind Systemelemente
 - Softwaresystem: System dessen Systemkomponenten und Systemelementen aus Software
-
- Informationstheorie: Übertragung und Kodierung von Informationen in Daten
 - Information: Wissen das Absender einem Empfänger über Kanal vermittelt, Kanal überträgt Daten die durch Zeichen kodiert sind, Zeichen werden durch Syntax zu Daten angeordnet, Durch Interpretationsvorschrift wird aus Daten Information
-
- Cybersicherheit: Alle Aspekte IT-Sicherheit mit Aktionsfeld auf gesamten Cyberraum (Nicht nur System)
 - Cyberraum: Sämtliche mit dem Internet verbundene IT-Systeme und IT-Infrastrukturen mit Kommunikation, Anwendungen, Prozesse mit Daten, Informationen, Wissen, Intelligenzen und Akteure
 - Informationssicherheit: Zielt auf Schutz von Informationen in allen Formen durch technische, personelle und organisatorische Maßnahmen



- Teilgebiete:
 - Kryptologie: Kryptographie (Wissenschaft d. Verschlüsselungsverfahren) + Kryptoanalyse (Wissenschaft d. Analyse von Methoden d. Kryptographie => Ziel: Unbefugtes Lesen / Manipulieren v. Informationen) = Erzeugen v. widerstandsfähigen Informationssystemen + Analyse der Schutzmechanismen
 - IT-Forensik: Methodische Datenanalyse zur Aufklärung von Sicherheitsvorfällen in IT-Systemen
 - Anwendungssicherheit: Schutz von Anwendungen gegen Angriffe
 - Systemsicherheit: Schutz von Systemen gegen Angriffe
 - Netz- / Kommunikationssicherheit: Schutz von Kommunikationssystemen gegen Angriffe
- Security-Engineering: Bereich der IT-Sicherheit -> Techniken, Werkzeuge, Prozessen und Methoden für den Entwurf, Implementierung, Test, Anpassung (existierender) von Systemen (an sich ändernde Umweltbedingungen) mit dem Ziel, Systeme zu erzeugen, die unter Angriffen zuverlässig funktionieren



Security Engineering im Softwarelebenszyklus



2. Grundlagen

Symmetrische Verschlüsselung

- Steganographie: Einbetten von Informationen in andere Informationen ohne dass diese entdeckt werden, Selbst bei Erkennung sollte eingebettete Information nicht einfach extrahierbar sein aus der Hülle, Eingebettete Information sollte robust gegen Änderung an Hülle sein (z.B. Nachricht in Bild)
 - Steganographie-Schlüssel -> Welche Pixel beinhalten Informationen, Wo ist die Nachricht versteckt
- Codierung: Ändert Bedeutung / Darstellung aber keine Geheimhaltung
- Alte Verfahren: Transposition (z.B. Schablone), Substitution (z.B. Cäsar, Vigenere-Chiffre)

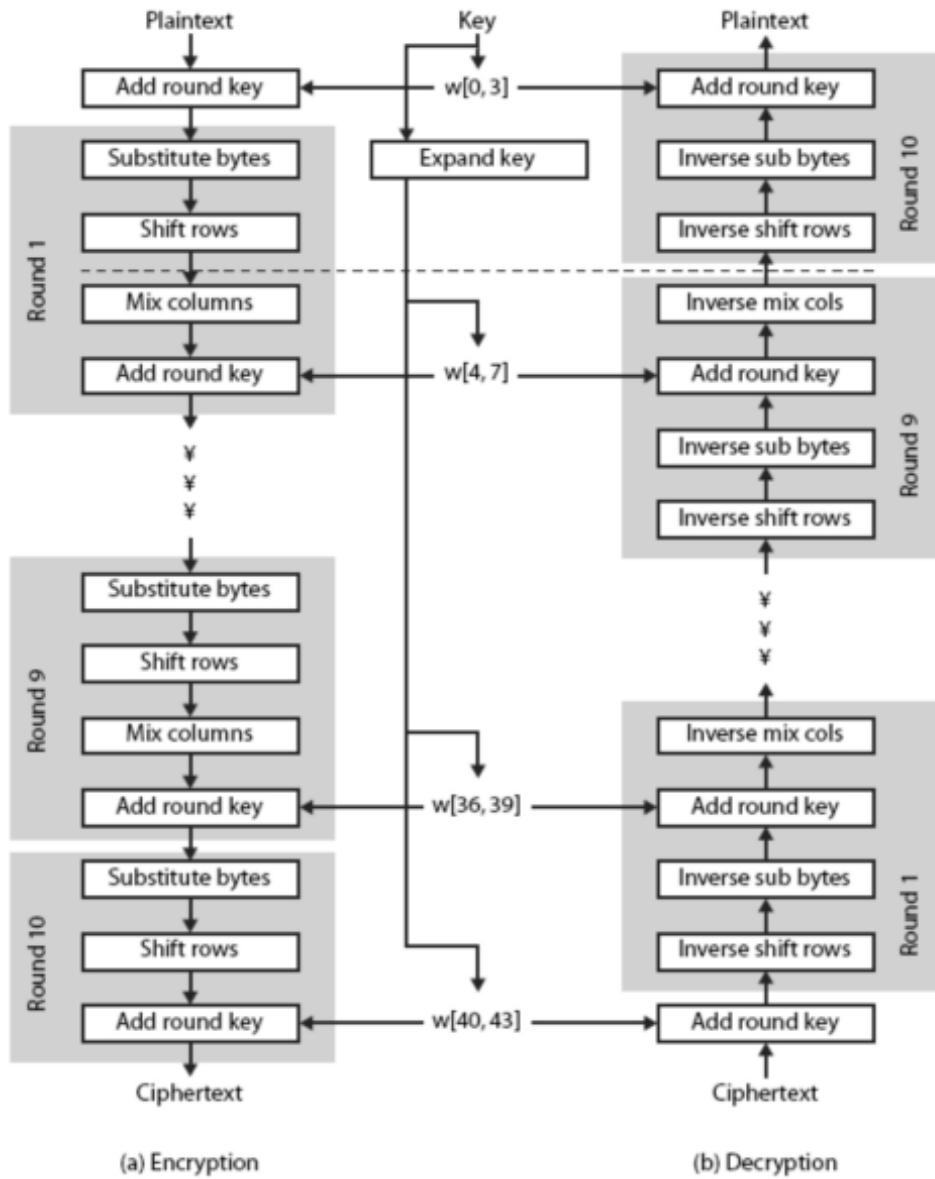
-
- Kryptosystem:
 - P: Menge aller zulässigen Klartexte (Plaintexts)
 - C: Menge aller Geheimtexte (Chiffre, Ciphertexts)
 - K: Menge aller Schlüssel (Keys)
 - $E = \{E_k : k \in K\}$: Menge aller Verschlüsselungsfunktionen, $E_k : P \rightarrow C$ Klartexte in Geheimtexte überführt
 - $D = \{D_k : k \in K\}$: Menge aller Entschlüsselungsfunktionen, $D_k : C \rightarrow P$ Geheimtexte in Klartexte überführt
 - Symmetrisch da: $D_k(E_k(p)) = p$
 - Chiffre = Symmetrischer Verschlüsselungsalgorithmus vs. Chiffrat = Verschlüsselter Text
 - Chiffrat = $E_k(\text{Klartext})$ vs. Klartext = $D_k(\text{Chiffrat})$
 - Weitere Definition Kryptosystem: Sammlung von Kryptografische Algorithmen zusammen mit Schlüsselverwaltungsprozessen um diese Nutzen zu können

-
- Kerkhoffs Prinzip: Sicherheit v. Verschlüsselungsverfahren darf nur auf Geheimhaltung d. Schlüssels beruhen, nicht auf der Geheimhaltung d. Verschlüsselungsalgorithmus -> Gilt für alle

Verschlüsselungsverfahren

- Problem: Schlüsselaustausch, Mehrere Personen (Gruppenschlüssel)
-

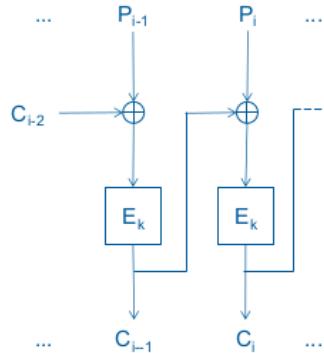
- AES (Advanced Encryption Standard -> Nachfolger von DES)
 - <https://www.cryptool.org/en/cto/aes-animation>
 - Mathematik:
 - Körper (A, \oplus, \odot) = Algebraische Struktur die Addition und Multiplikation zulässt (Subtraktion und Division / Inversion auch möglich)
 - (A, \oplus) und $(A \setminus \{0\}, \odot)$ sind kommutative Gruppen und es gilt Distributivgesetz
 - Falls $|A| < \infty$ dann endlicher Körper / Galois-Feld
 - Es gibt einen endlichen Körper $|A| = n$ wenn n eine Primzahlpotenz ist, $n = p^k$ für eine Primzahl p
 - Darstellung der Elemente von $GF(p^k)$ als k -Tupel über \mathbb{Z}_p , z.B. $(2, 0, 3, 4)$ aus $GF(5^4) = 2x^3 + 3x + 4$
 - Daher: Addition = XOR
 - Ablauf einer Runde:
 - Initialrunde: AddRoundKey
 - SubBytes: Byteweise Substitution
 - S-Box invertierbare Abbildung -> Jedes Element auf anderes Element, Nicht linear -> Schutz gegen differentielle und lineare Kryptoanalyse, da nur schwer Rückschlüsse auf Schlüssel oder Daten zu treffen sind (Input-Output wurde nicht nur linear transformiert (Output kann als lineare Kombination des Inputs dargestellt werden))
 - z.B. aus $B_0 = 67$ wird 85
 - ShiftRows: Zyklischer Links-Shift (Zeile 1: 0, Zeile 2: 1, ...)
 - MixColumns: Spaltenweise Multiplikation mit Polynom, Jedes Byte der Spalte wird mit jedem anderen der Spalte verknüpft (Außer in letzter Runde sowohl bei Encode / Decode)
 - Funktioniert mithilfe einer Matrix C die für die Verteilung auf die Spalte sorgt (Eine Änderung an Eingabe führt zu Änderung an 4 Ausgaben)
 - Linear und invertierbar
 - AddRoundKey: Mit Rundenschlüssel verknüpfen (XOR) -> Dadurch ist dieser Schritt selbstinvers
 - Rundenschlüssel abgeleitet durch Schlüssel (Key Expansion)
 - Anzahl Rundenschlüssel = 1 + Anzahl Runden
 - Eigenschaften:
 - Teile zu Kennen bringt nichts
 - Bits des Schlüssels in allen Rundenschlüsseln einbezogen
 - Genügend Nichtlinearität um Analyse zu erschweren



- Implementierung auf 8-Bit CPUs möglich mithilfe von Lookup Tabelle und Byte XOR
- Blockmodus: Legt fest wie Abfolge von Blöcken handzuhaben ist
 - Wichtig:
 - Vertraulichkeit: Information nur für bestimmten Empfängerkreis
 - Authentizität: Echtheit -> Zuordnung Datenursprung / Sender (Signaturen)
 - Integrität: Erkennung von Modifikationen
 - ECB: Electronic Codebook Mode -> Jeder Block einzeln, Keine Verknüpfung daher kein Schutz
 - Selbe Klartextblöcke geben selben Cipherblock -> Muster erkennbar
 - CBC: Cipher Block Chaining -> Chiffrat des vorherigen Blocks in Verschlüsselung des aktuellen
 - Abhängigkeit zwischen Blöcken (Keine Vertauschung) -> Um mittleren Block zu entschlüsseln alle vorherigen
 - Aber: Nicht parallelisierbar, Bitfehler Chiffrat multiplizieren sich beim Entschlüsseln im Klartext

CBC

Grafisch:



C_i = i-ter verschlüsselter Block
 P_i = i-ter Klartextblock
 C = Initialisierungsvektor (IV)
 \oplus = logisches XOR (Exklusives Oder)

IV entweder Zufallszahlen oder Zeitstempel

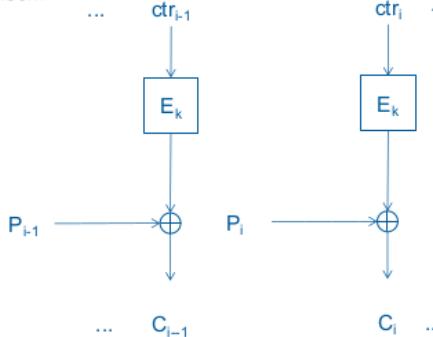
Formal: $C_i = E_k(P_i \oplus C_{i-1})$

Frage: Wie sieht die Entschlüsselung aus?

- CTR: Counter Mode -> Block wird mit Zähler verknüpft, Reihenfolge ebenfalls sicher

CTR

Grafisch:



C_i Block = i-ter verschlüsselter Block
 P_i = i-ter Klartextblock
 Ctr_i = i-ter Counterwert
 \oplus = logisches XOR (Exklusives Oder)

Counterwert besteht aus Nonce + Zähler

Formal: $C_i = P_i \oplus E_k(ctr_i)$

Frage: Wie sieht die Entschlüsselung aus?

- Warum Counter Verschlüsseln?

- Bitfehler beschädigt nur den jeweiligen Bit des Klartexts (Nicht den Block oder weitere)
- Verschlüsseln und Entschlüsseln identisch
- Vorberechnung Schlüsselstrom (Parallel, Wahlfreier Block)
- Da Counter hochzählt keine gute Entropie -> Ähnliche Blöcke geben Rückschlüsse auf Counter da XOR
- CCM: Counter mit CBC-MAC -> Zusätzlich Authentifizierung und Integrität neben Vertraulichkeit
- GCM: Galois/Counter Mode -> Zusätzlich Authentifizierung und Integrität neben Vertraulichkeit

Asymmetrische Verschlüsselung

- Problem: Sichere Schlüsselübertragung bei symmetrischer Verschlüsselung -> Lösung: Asymmetrische Verschlüsselung
- Öffentlicher Schlüssel (Public Key) -> Bekanntgabe, Privater Schlüssel (Geheim)

- Verschlüsseln mit öffentlichem Schlüssel des Empfängers
- Entschlüsseln mit eigenem privaten Schlüssel
- Signieren einer Nachricht mit eigenem privaten Schlüssel -> Signatur
- Prüfen der Signatur mit öffentlichen Schlüssel -> Prüfung Identität
- Bsp. RSA, ECIES (Elliptic Curve Integrated Encryption Scheme), DLIES (Discrete Logarithm Integrated Encryption Scheme)
- Basiert auf Problem: Faktorisierung, Berechnung diskreter Logarithmus, Berechnung quadratischen Resten
- Diskreter Logarithmus Problem:
 - Primzahl p , Erzeugendes Element α aus \mathbb{Z}_p^* , Element β aus \mathbb{Z}_p^*
 - Erzeugendes Element: Durch Potenzierung alle Elemente aus \mathbb{Z}_p^* erzeugbar (Auf Ring von P)
 - Gesucht: Eindeutige ganze Zahl a , $1 \leq a \leq p-1$, so dass $\alpha^a \equiv \beta \pmod{p}$
 - Funktioniert da: Wenn Exponent bekannt Ergebnis leicht ausrechenbar; Die Suche jedoch ohne Quantencomputer aktuell kein effizienter Algorithmus
- RSA:
 - Faktorisierungsproblem (Schwierigkeit die originalen Primzahlen zu finden, nachdem sie multipliziert wurden)
 - Primzahl p u. q mit $p \neq q$ (Nicht nah beieinander)
 - $n = p * q$
 - Klartextmenge / Ciphertextmenge \mathbb{Z}_n
 - Schlüssel $k = (n, p, q, e, d)$ wobei e so dass $ed \equiv 1 \pmod{\phi(n)}$
 - $\phi(n) = (p-1)(q-1)$
 - Öffentlich: $\text{pub} = (n, e)$
 - Privat: $\text{priv} = (p, q, d)$
 - Zusammen: Schlüsselwertpaar
 - Verschlüsselung: $E_{\text{pub}}(x) = x^e \pmod{n}$ ($x \in \mathbb{Z}_n$)
 - Entschlüsselung: $D_{\text{priv}}(y) = y^d \pmod{n}$

Zusatz

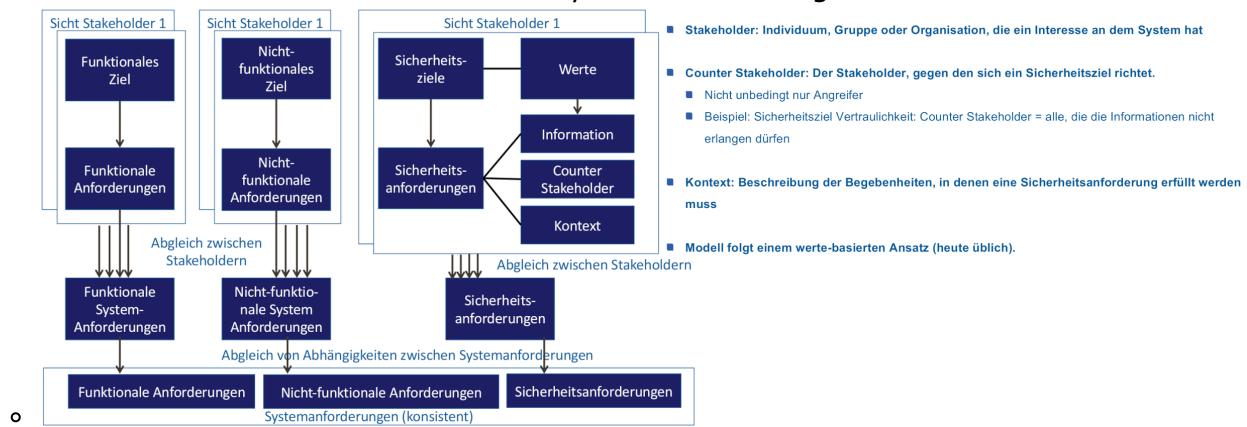
- Key-Derivation-Function: Kryptografischer Algorithmus zur Ableitung eines oder mehrerer Schlüssel von einem geheim Wert (z.B. Master Passwort)
- Key-Expansion-Function: Ableitung mehrerer Rundenschlüssel von einem geheimen Schlüssel
- Diffie-Hellman key exchange:
 - Ziel: Sicherer Austausch eines Symmetrischen Schlüssels
 - Einigung auf öffentliche große Primzahl p und natürliche Zahl g (Erzeuger zyklischer Gruppe) die kleiner als p
 - Alice, Bob erzeugen geheime Zufallszahl a bzw. b aus Menge $\{1, \dots, p-1\}$
 - Berechnung öffentlicher Schlüssel mit geheim Zahl und Verschicken an Partner
 - Mit privaten Schlüssel und öffentlichen Schlüssel kann gemeinsamer Schlüssel K abgeleitet werden
- Gruppen Schlüssel: Symmetrischer Schlüssel der Verschlüsselung innerhalb einer Gruppe von Benutzern ermöglicht (z.B. iPhones) (Geheim Übertragung notwendig / Ab Werk)

3. Security Engineering

Anforderungsanalyse / Requirements Engineering

- Anforderung: Spezifiert eine zu erfüllende Eigenschaft / Leistung eines Produktes, Systems oder Prozesses
 - Welche Eigenschaften: Vollständig, Kontextfrei (Klar verständlich), Konsistent (Keine Widersprüche), Eindeutig, Relevant (Notwendig), Atomar (Kurz und knapp), Testbar, Priorisierbar, ...
 - Akzeptanzkriterium: Fachliche Anforderung die zum Zeitpunkt der Abnahme erfüllt sein muss -> Akzeptanztests
 - Funktionale Anforderungen: Direkter Bezug zur Zweckbestimmung des Produkts, Was soll das Produkt tun?
 - Nicht-Funktionale Anforderungen: Unspezifisch, Wie soll das Produkt die Aufgaben tun?, z.B. Zeitverhalten, Ressourcenverbrauch, Robustheit, IT-Sicherheit, ...
 - Dokumentation: Anforderungskatalog, Lastenheft
- Sicherheitsanforderungen (Problematik)
 - Sowohl funktionale oder nicht funktionale Anforderungen (meistens)
 - Funktionale Anforderungen wirken sich auf funktionale Anforderungen aus -> Erheblicher Mehraufwand bei späten Sicherheitsanforderungen
 - Akzeptanzkriterium bei nicht funktionalen Anforderungen schwierig zu definieren
 - "Verschwinden" oft in funktionalen Anforderungen -> Problem: Mangelnde Sichtbarkeit (Übersehen / Unterschätzen, Ignorieren, Rückverfolgbarkeit), Gefahr vor Sicherheitslücken, Mangelnde Prüfbarkeit, Fehlende Priorisierung
 - Umgang: Abbildung nicht-funktionale Eigenschaft auf funktionale Eigenschaft oder funktionale Eigenschaft + leicht zu überprüfende nicht-funktionale Eigenschaft, Keine Umwandlung möglich / Prüfung Eigenschaft nicht möglich -> Annahme, dass nach paar Prüfungen Akzeptanzkriterium erfüllt ist (Korrektheit evtl. nicht gültig)
 - Weitere Probleme: Erst spät erhoben -> Änderungen, Auftraggeber können nicht formulieren, Mangelnde Sensitivität, Abhängigkeiten oft nicht erkennbar
- Requirements Engineering: Prozess zur Ermittlung von Anforderungen an ein Softwareprodukt
 - Security Requirements Engineering: Prozess zur Ermittlung von Sicherheitsanforderungen im Rahmen des Requirements Engineering
 - Probleme: Realistische Annahmen für Bedrohungslage, Richtiges Sicherheitsniveau
 - Qualitätsanforderungen:
 - Gilt für Alle:
 - In Vorgaben / Entscheidungen der Teilhaber (Stakeholder) begründet
 - Eindeutig, Klar, Prägnant, Widerspruchsfrei
 - Nachprüfbar (Erfüllungsgrad)
 - Unterscheidbar (nicht-funktional, funktional, Sicherheitsanforderung)
 - Gilt für funktionale Anforderungen:
 - Orthogonal / unabhängig voneinander
 - Erfüllbar durch Funktion/Eigenschaft des Systems
 - Gilt für nicht-funktionalen Anforderungen:
 - Funktions- bzw. Servicequalität des Produkts / Systems
 - Explizit hinsichtlich des Erfüllungsgrades zu spezifizieren
 - Gilt für Sicherheitsanforderungen:
 - Zurückzuführen auf Sicherheitsziele / Sicherheitsbedürfnisse der Stakeholder und Nutzer oder Bedrohungen schützenswerter Güter oder Personen

- Restriktionen des technischen / technologischen Lösungsraum
- Risikoanalyse der zu schützenden Werte als Grundlage
- Orthogonal
- Erfüllbar durch Funktion / Eigenschaft des Systems / Systemumgebung
- Keine zweifelhafte Annahme / teilweise Erfüllung



Bedrohungs- und Risikoanalyse

- Strukturierte Vorgehensweise zur Ermittlung von Sicherheitsanforderungen unter Berücksichtigung Risiko -> Wirtschaftlichkeit

1. Analyse der schützenswerten Güter (=Werte)
2. Identifikation möglicher Bedrohungen für diese Werte
3. Bewertung des mit der Bedrohung verbundenen Risikos
 - o Gegenmaßnahmen basierend auf Risiko -> Für hohe/kritische Risiken -> Neue Anforderungen
 - o Überblick / 6-Phasen Modell

1. Phase: Werte, Akteure und Operationen identifizieren

- Werte = zu schützende Güter, Bedeutung von Werten unterschiedlich für Stakeholder (Individuelle Sicht)
- Akteure = Personen oder Prozesse, die Operationen auf Werten durchführen
- Operationen = Handlungen, die Akteure auf Werten durchführen
- Ziel: Konzentration auf Werte, Akteure, Operationen -> Reduktion Problemstellung -> Verhindert Over Engineering / Falscher Fokus

2. Phase: Überblick Architektur

- Bedrohungsanalyse erfordert grobe Architektur -> Existierendes System: Architektur liegt vor, Neues System: Henne-Ei Problem, Gute Architektur basiert auf Sicherheitsanforderungen
- Änderungen -> Bedrohungsmodellierung iterativer Prozess -> Sicherheitsvorgaben / Architektur iterativ verfeinern
- Da Werte meist Daten -> Fokus Architektur auf Datenhaltung und Datenflüsse
 - Problem Modellierung: UML kein Datenflussdiagramm
 - Elemente: Verarbeitungsfunktion (Process), Speicher/Datenbank (Data Store), Start/Endpunkt (Extern Entity), Datenflüsse (Flow)
 - z.B. Yourdon / DeMarco, Gane / Sarson, Yourdon / Coad

- Ziel: Identifikation wo Werte gespeichert sind in Architektur, Welche Systemkomponenten Datenflüsse mit Werten senden, Frühe Architekturskizze -> Konkrete Analyse -> Brauchbare Ergebnisse

3. Phase: System zerteilen

- **Vertrauengrenze:** Vertrauengrenze ist eine tatsächliche oder gedachte Abgrenzung zwischen zwei oder mehr Komponenten in einem System. Sie trennt im System zwei oder mehr Bereiche mit unterschiedlichem Sicherheitsniveau.
- **Identifikation und Dokumentation von Vertrauengrenzen (Trust Boundaries)**
 - Bereiche mit gleichem Sicherheitsniveau identifizieren
 - Grenzen identifizieren -> Datenfluss sehr kontrolliert oder gar nicht
 - Mehr Vertrauengrenzen (fein granular) -> Höhere Sicherheit / Mehr Aufwand
 - System entlang Vertrauengrenzen zerteilen -> Evtl. Phase 2 wiederholen (Granularität)
- **Ziel:** Zerteilung System aus Sicherheitssicht gibt Überblick über Sicherheits-Domänen -> Einsatz von Sicherheitsmaßnahmen / Ansatzpunkt für Bedrohungsanalyse an Vertrauengrenzen

4. Phase: Bedrohungen identifizieren und dokumentieren

- **Bedrohungen:** Umstand oder Ereignis, durch den oder das ein Schaden entstehen kann (Bezogen auf einen Wert). In Infomationstechnik: Bezogen auf Verfügbarkeit, Integrität, Vertraulichkeit -> Besitzer / Benutzer der Information kann Schaden entstehen -> Trifft Bedrohung auf Schwachstelle (technisch oder organisatorische Mängel), so entsteht Gefährdung
 - Angriffe sind Bedrohungen bzw. Gefährdungen für Systeme
- **Ziel:** Möglichst viele Bedrohungen für Werte aus Phase 1 identifizieren
- **Vorgehen:**
 - Bedrohungen pro Wert erheben (evtl. Werte-Gruppen) -> Welche Angriffe auf einen Wert möglich, Welche zusätzlichen Operationen/unberechtigten Operationen auf einem Wert möglich
 - Bedrohungen an Vertrauengrenzen erheben
- **Techniken:**
 - Klassifizierung STRIDE: Spoofing (Erschleichen Identität), Tampering (Verändern v. Informationen), Repudiation (Abstreiten v. Aktionen), Information Disclosure (Veröffentlichen v. Informationen), Denial of Service (Angriff auf Verfügbarkeit), Elevation of Privilege (Verschaffen von Rechten)
 - Brainstorming: Fragestellung: Wo angreifen, um den Wert xy zu schädigen? -> Gruppierung und Bewertung der Relevanz
 - Checklisten: Liste mit Angriffsarten, Schnelle Durchführung, Jeodch Vollständigkeit, Qualität, Eignung für Domänenmodell nicht sichergestellt -> Daher gute Ergänzung (z.b. OWASP Top 10)
 - Datenflussanalyse: Betrachtung Datenflussdiagramme aus Phase 3 -> Fokus auf Externen Datenflüssen / Datenflüsse über Vertrauengrenzen -> Problem: Blinder Fleck (Nur fließende Werte betrachtet)
 - Angriffsbaum: Wurzel = Grundwert, Verschiedene Bedrohungen (Kinder), Verfeinerung, Oder-Semantik (Explizit UND)
 - Misuse Case: UML Use Case (SCRUM) -> Macht Bedrohung für Use Cases des Produkts sichtbar
 - Misuser + Misuse Case (Aktion d. Angreifers)
 - <<threaten>>: Verbindung Misuse Case u. bedrohter UseCase

- <<mitigate>>: Verbindung Security Use Case und abgemildeter Misuse Case (Wirkt entgegen des Misuse Case)
 - Nach Identifikation: Bedrohungen bezüglich Relevanz bewertet und irrelevante fallengelassen, Pro Wert Liste von Bedrohungen die diesen gefährden (Werden weiter betrachtet)
5. Phase: Bedrohungen bewerten
- Fokussierter Einsatz von Ressourcen für Bewertung d. Bedrohungen
 - Bewertung von Risiko für einen Wert
 - Risk Management: Risiko = Eintrittswahrscheinlichkeit x Schadenshöhe
 - OWASP Risk Rating (0-9):
 - 4 Themengruppen:
 1. Threat Agent (Angreifer) -> Eintrittswahrscheinlichkeit
 - Skill Level: 1 (Kein Können), 5 (Erfahrener Computerbenutzer), 9 (Security Penetration Tester)
 - Motive: 1 (Kein Nutzen / Belohnung), 4 (Evtl. Nutzen / ""), 9 (Großer Nutzen / "")
 - Opportunity: 1 (Voller Zugriff / Viele Ressourcen erforderlich), 9 (Kein(e) Zugriff / Ressourcen erforderlich)
 - Size (Gruppe d. Angreifer): 2 (Developers / Admins), 6 (Authentifizierte Users), 9 (Anonyme Internet Users)
 2. Vulnerability (Schwachstelle) -> Eintrittswahrscheinlichkeit
 - Ease of Discovery (Erkennung): 1 ("Unmöglich"), 3 (Schwer), 9 (Automatische Tools verfügbar)
 - Ease of Exploit (Ausnutzen): 1 (Theoretisch), 3 (Schwer), 9 (Automatische Tools verfügbar)
 - Awareness (Bekanntheit): 1 (Unbekannt), 4 (Versteckt), 9 (Öffentlich bekannt)
 - Intrusion Detection (Erkennung d. Angriffs): 1 (IDS aktiv), 3 (Logging + Untersuchung), 9 (Kein Logging)
 3. Technical Impact (Technischer Schaden) -> Schadenshöhe
 - Loss of Confidentiality (Wie viele Daten / wie sensitive ... entwendet)
 - Loss of Integrity (Wie viele Daten beschädigt)
 - Loss of Availability (Downtime / Wichtigkeit von Services)
 - Loss of Accountability (Sind Attacken auf Individuum zurückzuführen)
 4. Business Impact (Wirtschaftlicher Schaden) -> Schadenshöhe
 - Financial damage: 1 (Günstiger als Behebung d. Problems), 7 (Schaden bemerkbar am Jahresumsatz)
 - Reputation damage (Rufschaden): 5 (Tausende Menschen), 9 (Millionen Menschen)
 - Non-compliance (Führt zur Nichteinhaltung von Verträgen / Zertifikaten / Vereinbarungen)
 - Privacy violation (Wie viele privat identifizierbare Daten geklaut): 5 (Tausende Menschen), 9 (Millionen Menschen)

- 0 < 3: LOW, 3 < 6 MEDIUM, 6 <= 9 HIGH -> Durchschnitt bilden und in Matrix sortieren

- => Umgang mit Risiko nun möglich => Akzeptieren, Vermeiden, Abmildern

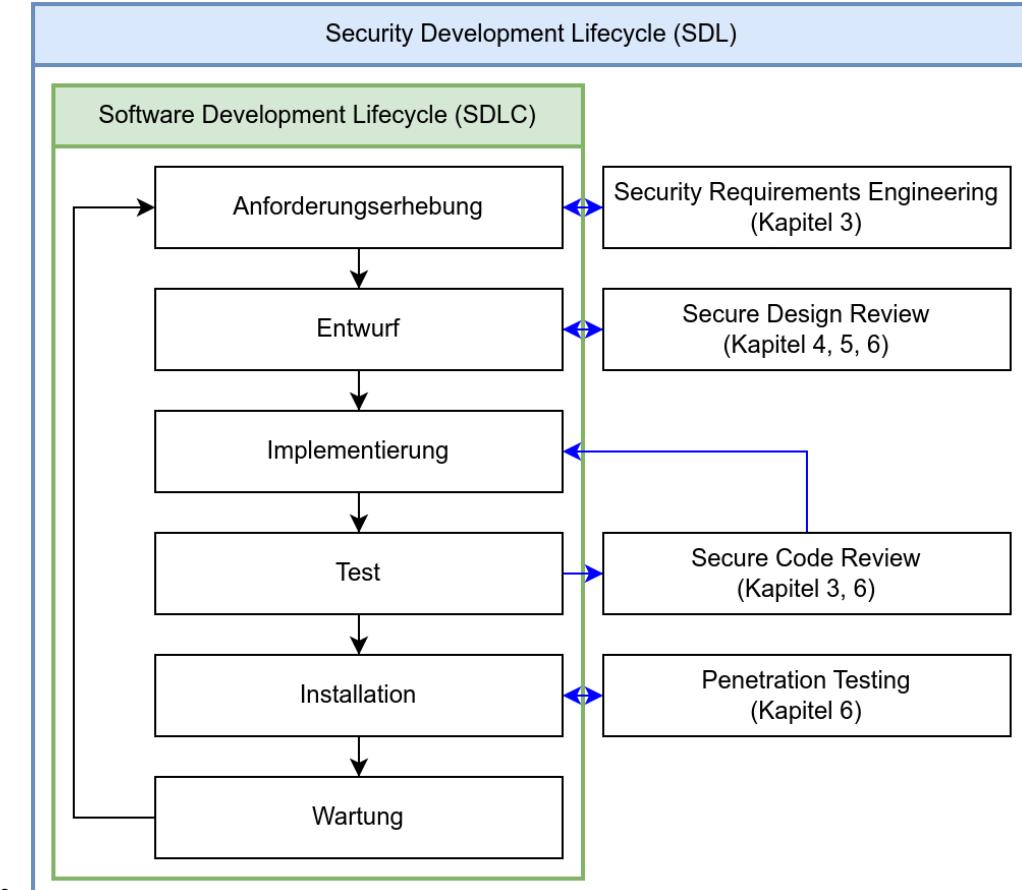
6. Phase: Gegenmaßnahmen planen

- Angriff nach Kritikalität ordnen (Gesamtrisiko) -> Abwägen anhand Budget
- Fragen: Warum ist Angriff möglich? Was ist das Problem? Gewünschte Funktionalität das Problem oder logischer Fehler im Sicherheitskonzept?, Welche Änderung der Architektur notwendig? Dadurch neue Schwachstelle?, Welche Angriffe betreffen selbe Schwachstelle? Abwehr von mehreren Angriffen möglich?, Vertrauensgrenzen neu ziehen?
- Formulierung neuer Sicherheitsanforderungen um hohe und kritische Risiken zu behandeln
- Problem: Betrachtet nur bekannte Bedrohungen, Motiviert dazu nur das Nötigste zu machen, Risikobewertung Domänenpezifisch, Kette von Vorfällen nicht betrachtet

- Modellbasierte Risikoanalyse CORAS (Speziell für sicherheitskritische Systeme) -> Siehe Anhang

Security Development Lifecycle (SDL)

- Strukturierter Prozess = Kontrolle von Kosten, Terminen, Qualität
 - IT-Sicherheit Qualitätsaspekt -> Brücksichtung in Softwareentwicklungsprozess
 - Ziel: Vorgehensmodell zur Entwicklung sicherer Software

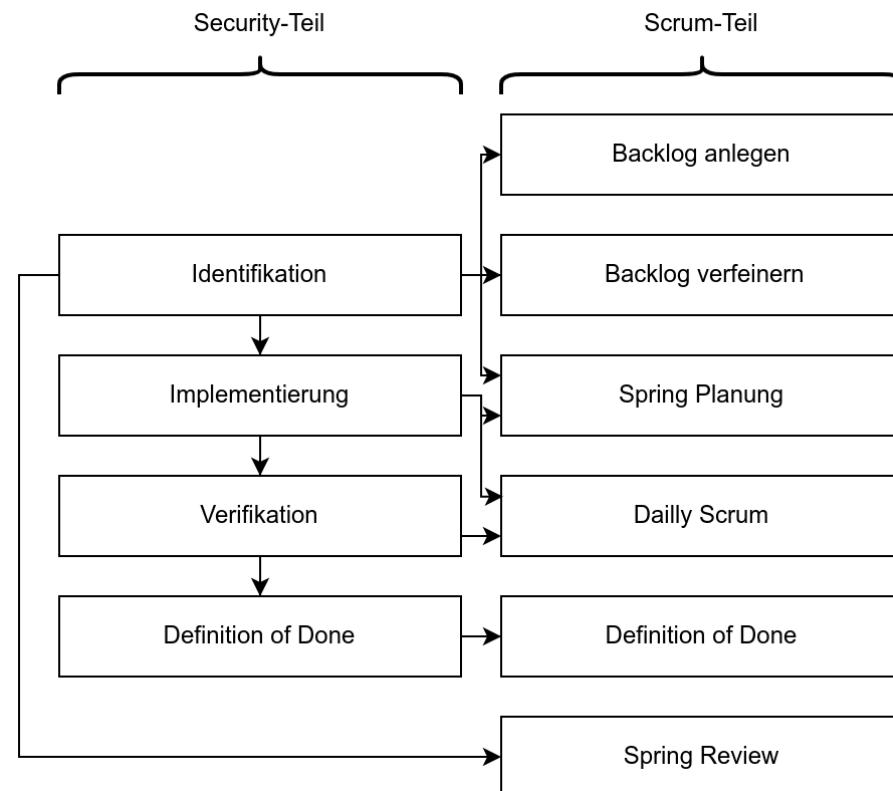


- Microsoft SDL

1. Training: Vermittlung Grundlagen (Secure Design, Bedrohungsmodellierung, Secure Programming, Testen von Sicherheit, Best Practices Datenschutz)

2. Requirements: Sicherheits- Datenschutzanforderungen erheben, Quality Gates (Minimalanforderungen) / Bug Bars definieren (Welche Verwundbarkeiten erlaubt), Risikoanalyse
3. Design: Design-spezifische Sicherheitsanforderungen (z.B. Minimalanforderungen an genutzte Krypto), Analyse / Reduktion Angriffsfläche (siehe Kapitel 4 Prinzipien), Bedrohungsmodellierung
4. Implementation: Erlaubte Werkzeuge + Sicherheitsmaßnahmen (z.B. Compiler-Warnungen), Verbot unsicherer Funktionen (APIs) + Code um auf die Durchsetzung zu prüfen, Statische Codeanalyse
5. Verification: Dynamische Codeanalyse (Fokus Sicherheitsprobleme wie z.B. Speicher), Fuzzing (Fehlererzeugung), Überprüfung Angriffsfläche
6. Release: Incidence Response Plan (Vorgehen bei Bedrohung), Finales Security Review (Nutzt alles vorhergegangene), Zertifizierung / Archivierung (Einhaltung Sicherheit/Datenschutz zertifizieren + Sämtliche Projektdaten (Source Code, Doku, Binaries) geeignet archivieren)
7. Response: Incidence Response Plan ausführen + Software Updates
 - In agiler Projektwelt:
 - Jede Iteration: Bedrohungsmodellierung, Alle Schritte aus Implementation
 - Erste Iteration: Sicherheits- Datenschutzanforderungen erheben, Risikoanalyse, Design-spezifische Sicherheitsanforderungen, Analyse / Reduktion Angriffsfläche, Incidence Response Plan anlegen
 - Regelmäßig, mehrere Sprints: Quality Gates / Bug Bars definieren, Alle Schritte aus Verification
 - Grundsätze:
 - Secure by design: Schon in Planungsphase sollte auf die Sicherheitsbelange der Software eingegangen werden
 - Secure by default: Trotz sorgfältigster Planung sollte Entwickler von dem Vorhandensein von Sicherheitslücken ausgehen, Aus diesem Grund sollten Standardeinstellungen (z.B. Privilegien) möglichst niedrig gewählt werden und selten benutzte Features standardmäßig deaktiviert werden
 - Secure in deployment: Mitgelieferten Dokumentationen und Tools sollen die Administratoren dabei unterstützen die Software möglichst optimal einzurichten
 - Communications (Software): Entwickler sollten offen mit möglichen Sicherheitslücken umgehen und den Endanwendern schnell Patches oder Workarounds zur Verfügung stellen
 - Privacy by design: Schon in Planungsphase sollten Datenschutzbelange der Software berücksichtigt werden
 - Privacy by default: Standardeinstellungen der Software sollten konservativ gewählt werden
 - Privacy in deployment: Datenschutzmechanismen sollten offengelegt werden, um Administratoren zu ermöglichen, die internen Datenschutzrichtlinien des Unternehmens umzusetzen
 - Communications (Privacy): Datenschutzerklärungen sollten transparent formuliert werden, Ein Team für Datenschutzvorfälle sollte eingerichtet werden
- DevSecOps

- Zusammenarbeit versch. Team mit Ziel schnell IT-Dienste zur Verfügung stellen zu können
 - Qualitätssicherung, IT-Sicherheit, Entwickler, Betrieb
 - Leichtgewichtige / automatisierte Security Prozesse, Micorservices, Containerisierung, CI/CD
- Secure Scrum
 - SDL für Scrum
 - Keine neuen Rollen, Macht Sicherheitrelevanz sichtbar/verfolgbar



1. Identifikation (von sicherheitsrelevanten Punkten):

- Bei Backlog anlegen, Sprint Planung und Backlog Verfeinerung
- Durchgeführt durch Team, Owner, Customer
- Vorgehen:
 - Ranking User Storys (sicherheitsrelevant / nicht)
 - Aufstellen v. Misuse Cases und Ranking nach Risiko
 - Dokumentation durch S-Marks (Markierung) und S-Tags (Beschreibung) -> Security Cases

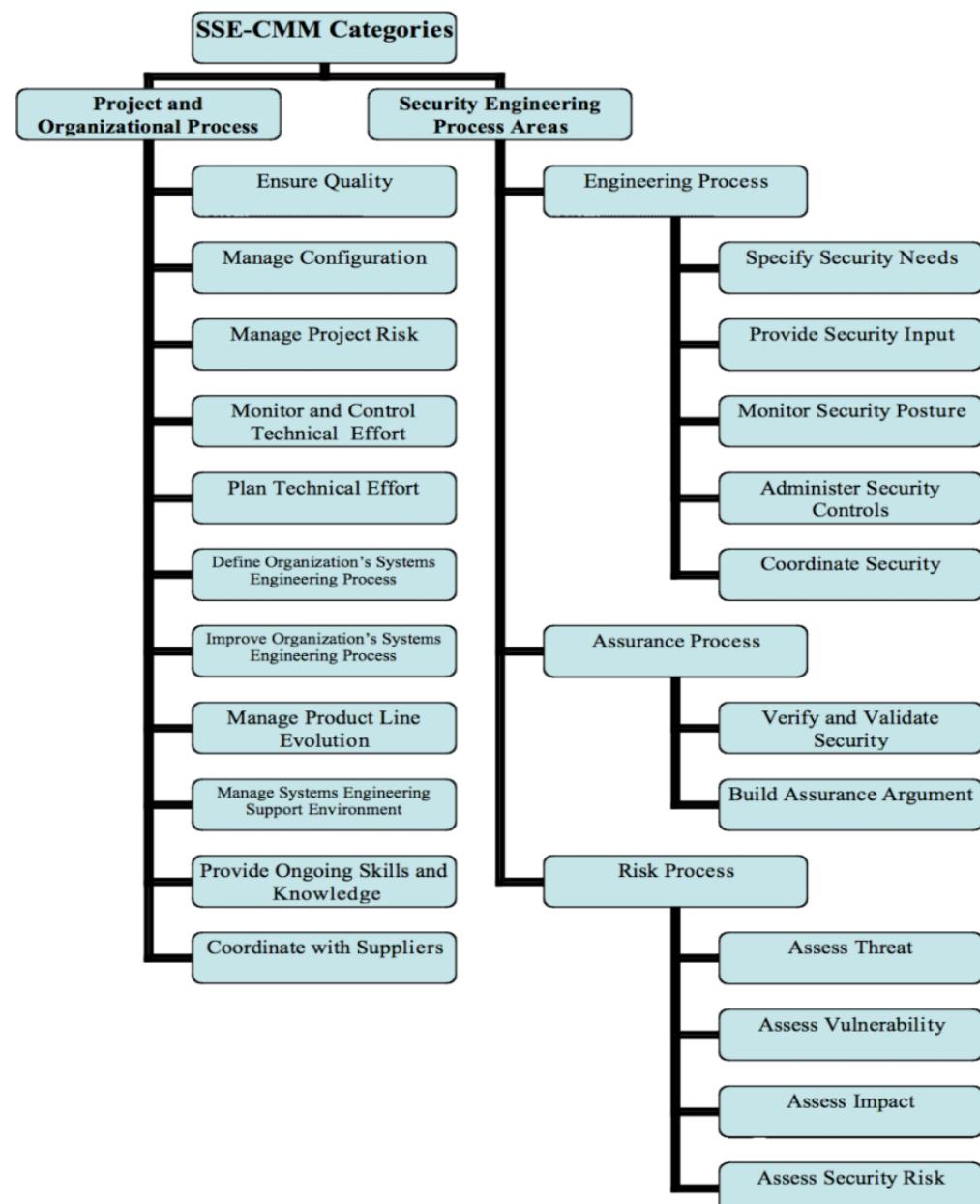
2. Implementierung:

- S-Tags im Backlog -> Sicherheitsrelevanz sichtbar
- Aufgabe: Teil einer User Story, der in einem Sprint entwickelt werden kann
- Verbindung aller Aufgaben, die mit sicherheitskritischen Use Cases zusammenhängen mit S-Tag
- S-Tag ebenfalls in Aufgaben aufteilbar, Mit ursprünglichen S-Tag verbunden

3. Verifikation / Definition of Done:

- Entwickler kann Ergebnis einer Aufgabe selbst / in angemessener Zeit verifizieren -> Verifikation der Definition of Done
 - Definition of Done = Vereinbarung, was erfüllt sein muss sodass User Story als fertig zählt

- Entwickler kann Ergebnis nicht selbst / in angemessener Zeit verifizieren -> Erzeuge Verifikationsaufgabe im Backlog, Verlinke mit S-Tag zur Story, Verifikation nicht Teil der DoD, Verifikation im nächsten Sprint
- Problem: Big-Bang Einführung eines SDL schwierig -> Inkrementelle Einführung -> Reifegradmodelle
 - OpenSAMM (Open Software Assurance Maturity Model)
 - 4 Business Functions mit je 3 Security Practices mit versch. Levels -> Aktivität pro Level (Inkrementelle Einführung)
 - Governance (Führung):
 - Strategy & Metrics: Umsetzung von OpenSAMM
 - Policy & Compliance: Übereinstimmung mit Recht, Gesetz u. verpflichtenden Standards u. Einführung / Überwachung interner Sicherheitsstandards
 - Education & Guidance: Verbesserung Security Knowhow d. Personals
 - Construction:
 - Threat Assessment: Identifikation / Verständnis von projektbezogenen Risiken
 - Security Requirements: Spezifikation der IT-Sicherheit
 - Secure Architecture: Sicheres Design / vorbeugende Sicherheitsmaßnahmen
 - Verification:
 - Design Review: Überprüfung des Designs und der Architektur aus Sicherheitssicht
 - Code Review: Inspektion des Source Codes
 - Security Testing: Untersuchung der ausführbaren Software
 - Deployment:
 - Vulnerability Management: Prozess zum Umgang mit erkannten Sicherheitsschwachstellen
 - Environment Hardening: Absicherung der Umgebung um Angriffe auf die entwickelte Software zuerschweren
 - Operational Enablement: Sammlung von sicherheitskritischen Informationen vom Projektteam und Kommunikation dieser Informationen an Benutzer und Betreiber
 - Überprüfung: Punkte für Security Practice festgehalten auf Score Cards für Überprüfung Fortschritt
 - SSE-CMM (System Software Engineering Capability Maturity Model)
 - Bewertung Entwicklungsprozess Security kann notwendig sein (Beurteilung)
 - SSE-CMM = Systematik zur Beschreibung wesentlicher Merkmale eines Sicherheitsentwicklungsprozesses
 - Empfehlung von generischen Aktivitäten / Basisaktivitäten in den Kategorien Security Engineering und Projekt u. Organisation



- Security Engineering unterteilt in:
 - Risiko
 - Vertrauenswürdigkeit
 - System-Lebenszyklus
- Prozess-Modell hat 2 Dimensionen -> Domäne (Spezifisch für Security Engineering und Projekt u. Organisation), Capability (Generisch, Anwendbar auf alle Prozesse, Reifegrad)
- Reifegrade:
 - 0: Nicht umgesetzt (Pro Stufe ca. 1 bis 3 Jahre (Start 2 bis 4))
 - 1: Formlos umgesetzt:
 - Vereinzelte Maßnahmen ohne Organisation
 - Situation: Unvorhersehbare Zeit, Kosten, Qualität
 - Notwendig: Planung, Überwachung, Änderungsmanagement, Qualitätssicherung
 - 2: Geplant u. verfolgt:
 - Stabiler Prozess existiert u. wird gelebt
 - Situation: Kosten / Qualität schwanken, Terminkontrolle

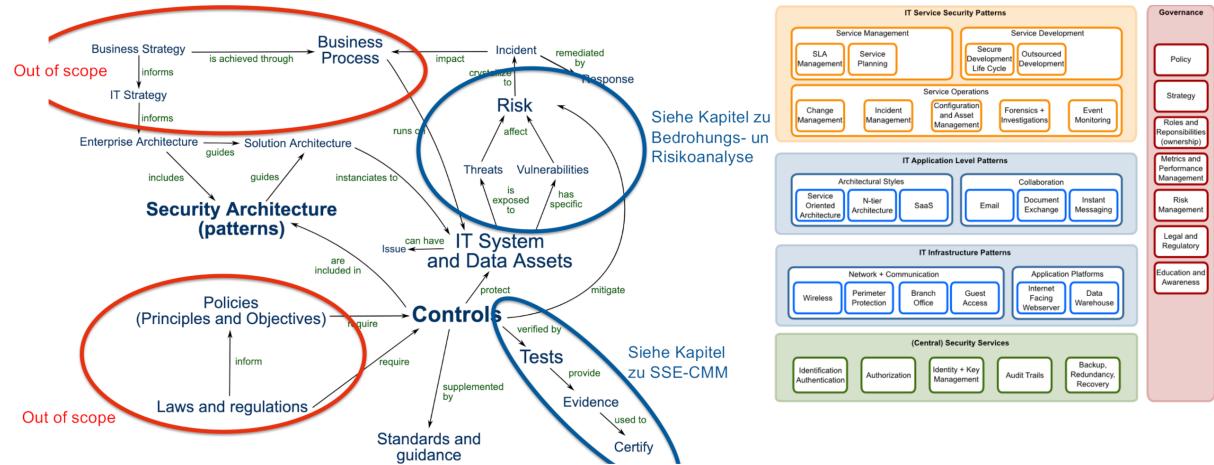
- Notwendig: Prozess-Standards entwickeln, Methoden einführen (Definition, Test, ...)
- 3: Gut definiert:
 - Prozess definiert, Prozessmodell existiert -> Konsistente Implementierung d. Prozess sichergestellt
 - Situation: Zuverlässige Kosten u. Termin, Verbesserte aber schwankende Qualität
 - Notwendig: Messen und Analysieren, Quantitative Qualitätssicherung
- 4: Quantitativ kontrolliert:
 - Prozessmessungen u. Analysen -> Weiterentwicklung d. Prozess genutzt
 - Situation: Gute statische Kontrolle d. Produktqualität
 - Notwendig: Prozessüberwachung, Quantitative Produktivitätspläne, Instrumentierte Prozessumgebung, Investition in Technologie, ...
- 5: Kontinuierlich verbessernd:
 - Management ist regelmäßig in Prozessbewertung u. Optimierung einbezogen
 - Situation: Quantitative Basis für Investitionen in Prozessautomatisierung u. Verbesserung
 - Notwendig: Kontinuierlich Prozessvermessung u. Prozessmethoden zur Fehlervermeidung
- 2 Dimensionenmodell (Verschiedene Prozesse und Handlungsaktivitäten sind für beide Bereiche definiert)
 - Capability: Sehr generalisiert und trifft auf mehrere Domains zu
 - Domain: Bezug auf Security Domain

4. Sicherheitsarchitektur

- Definition Sicherheitarchitektur (Schwerpunkt Informationssicherheit / Sicherheitsorganisation):
 - Bezogen auf Informationsverarbeitung
 - Gesamtheit aller in den Unternehmens- oder Behördenbereichen realisierten Sicherheitskonzepte und die daraus abgeleiteten Sicherheitsmaßnahmen (Informationssicherheit)
 - Maßnahmen in heterogenen Client/Server-Systemen sowie in Netzen und beim Anschluss an öffentliche Netze
 - Maßnahmen der Aufbau- und Ablauforganisation mit der gesamten sicherheitsrelevanten Dokumentation
 - Planungen sowie die Kontrollen auf Einhaltung der Konzepte, Richtlinien zum Einsatz der Maßnahmen selbst sowie der Schutz der Maßnahmen
 - Bereich wie: Strategie, Management, Infrastruktur, Software, Hardware, Zugriffskontrolle, Notfallvorsorge
- Definition Sicherheitarchitektur (Schwerpunkt Anwendungsentwicklung / System):
 - Beschreibt die grundlegenden Komponenten und deren Zusammenspiel innerhalb einer Software
 - Bündelung der gewonnenen Erkenntnisse zu Gesamtkonzept (Akteure mit ihrer Benutzung, Trust Boundaries, Identifizierte Assets, ...)

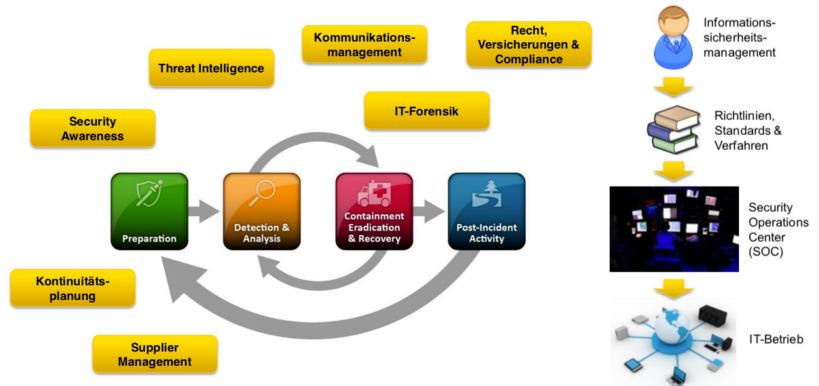
Open Security Architecture (OSA)

- Framework für Erstellung v. Sicherheitsarchitekturen
 - Bibliothek für Security Architecture Patterns + Security Controls
 - Definitionen:
 - Entwurfsmuster: Bewährte generische Lösung für ein wiederkehrendes Entwurfsproblem
 - Sicherheitsarchitekturmuster: Entwurfsmuster für Entwurfsproblem aus dem Bereich Sicherheitsarchitektur
 - Security Control: Schutz-/ Gegenmaßnahme die Sicherheitsziele (Integrität, Vertraulichkeit, Verfügbarkeit) schützt
 - Basierend auf Analyse der Umgebungsbedingung werden geeignete Security Controls ausgesucht und in angestrebte Architektur integriert
 - Security Controls:
 - Schutzmaßnahme (Schützen a priori), Gegenmaßnahmen (Begrenzen a posteriori den Schaden)
 - Standards angeboten von: NIST, ISO, BSI
 - Umfasst:



- Akteur: Ist eine prototypenmäßige Business-Rolle -> Visualisierung der verschiedenen Aufgaben und Verantwortungen die eine Rolle haben kann
 - End-User
 - Developer
 - IT-Security-Manger (Planning, policies, procedures, guidelines)
 - Infrastructure architect, software architect, Auditor, Product Owner, ...
- Sicherheitsprinzipien (WICHTIG!):
 - Architektur:
 - Simplicity over flexibility: Komplexität erschwert IT-Sicherheit -> Wunsch möglichst flexibel zu sein führt zu Komplexität -> KISS (small & simple) -> Reduziert meist Angriffsfläche, Anzahl Verwundbarkeiten, Möglichkeit Konfigurationsfehler
 - Usability over restriction: Benutzer haben Problem mit Sicherheitsmechanismen / Umgehen Sicherheitsmechanismen wenn aufwendig -> Auf Benutzbarkeit achten
 - Defence in depth: Früher Definition Perimeter + Schutz am Perimeter -> Deswegen Heute: Mehrere Sicherheitsschichten in Systemen mit unterschiedl. Funktionsprinzipien, unterschiedl. Herstellern, Angreifer muss Schichten sequentiell überwinden nicht nur eine Barriere

- Minimize Attack Surface: Teil eines Systems, der einem Angriff exponiert ist -> Angriffsfähigkeit minimieren, vollständig überwachen, Sicherheitsregeln durchsetzen, z.B. API Minimierung, Daten validierung / verifizierung, Login, ...
- Assume a state of compromise: Problem: Angreifer muss eine Sicherheitsschwachstelle finden vs. Verteidiger muss alle Sicherheitsschwachstellen kennen (Offense-Defense Balance Theorie) -> Daher: Annahme, dass Teile des Systems v. Angreifer kompromittiert sind / Netzwerk infiltriert -> Führt zu:
 - Schutz: Abschottung Netzwerke, (Interne) Kommunikation mit Authentizität / Vertraulichkeit, Integrität realisiert, Interne Server selber Schutz wie externe
 - Überwachung:
 - Security Operations Centers (Sicherheitsleitstelle)
 - Intrusion Detection / Prevention Systems:
 - Passiv, Aktiv
 - Hostbasiert / Netzwerkbasiert
 - Signaturbasierte Verfahren (Pattern Matching, Angriffsmuster)
-> Zuverlässig, Erkennt nur bekanntes
 - Anomalie-Erkennung (Normalverhalten anlernen -> Abweichung oberhalb Schranke als Angriff gewertet) -> Viele Fehlalarme, Erkennung neuer Angriffe möglich, ggf. Neues Anlernen nötig
 - Integritätsprüfung (Kryptografische Hashes v. Daten + Sichere Speicherung u. Vergleich) -> Schnell, Zuverlässig, Aufwand bei gewollten Änderungen (Updates)
 - SIEM (Security Information and Event Management)
 - Zentralisierung und Vereinfachung der Auswertung von Sicherheitsvorfällen / Umfassender Überblick über Sicherheitslage
 - Datensammlung (IDS / IPS, Firewalls, Anti virus, ...), Aggregation / Filterung, Normalisierung (Umwandlung in gemeinsame Semantik / Syntax), Korrelation (Versuch, Vorgehen des Angreifers nachzuverfolgen basierend auf Daten und zeitlicher Reihenfolge -> Erfordert Identitätsfeststellung des Angriffs), Berichterzeugung / Verfolgung, Archivierung
 - Ziel: Schnelle Erkennung, Vollständige Analyse, Management, Schadensminimierung, Wiederherstellung, Ursachen / Beweissicherung, Verhinderung Wiederholung, Verbesserung Vorfallsbehandlung



- Be reluctant to trust: Grundsätzlich von feindlicher Umgebung ausgehen, Vorsicht vor transitivem Vertrauen -> Vorsicht / Überprüfung bei Übergabe v. Daten, Zugriffe authentifizieren

- Steuerung Vertrauen z.B. Public Key Infrastructure mit Zertifikaten (Root-Zertifikate als Wurzel d. Vertrauens), Achtung: Transitives Vertrauen wie Web of Trust (Nicht Zertifizierungsstelle sondern Benutzer zertifizieren)
- Auf Anwendungsebene: Isolation oft bei Code aus unterschiedl. Quellen -> Kommunikation über API / Speicher, Isolation geht oft mit Virtualisierung einher (z.B. JVM, Container, ...)
- Auch wichtig: Code-Authentifizierung über Code Signatur -> Abgeleitet daraus: Berechtigungen d. Codes, Regeln für Zusammenspiel, ...

- Secure the weakest link: System nur so sicher wie die unsicherste ausnutzbare Komponente des Systems

- Vollständige, systemweite Bedrohungsanalyse notwendig

- Implementierung:

- Open design:
- Secure coding practices:
- Black box and white box testing:

- Betrieb und Konfiguration:

- Complete mediation: Problem: Race Conditions / Time-of-check-Time-of-use -> Daher:

- Jeder Zugriff abfangen, Berechtigung und Security Policy prüfen (Nicht cachen)
- Prüfende Komponente Reference Monitor und besonders geschützt / sorgfältig zu implementieren

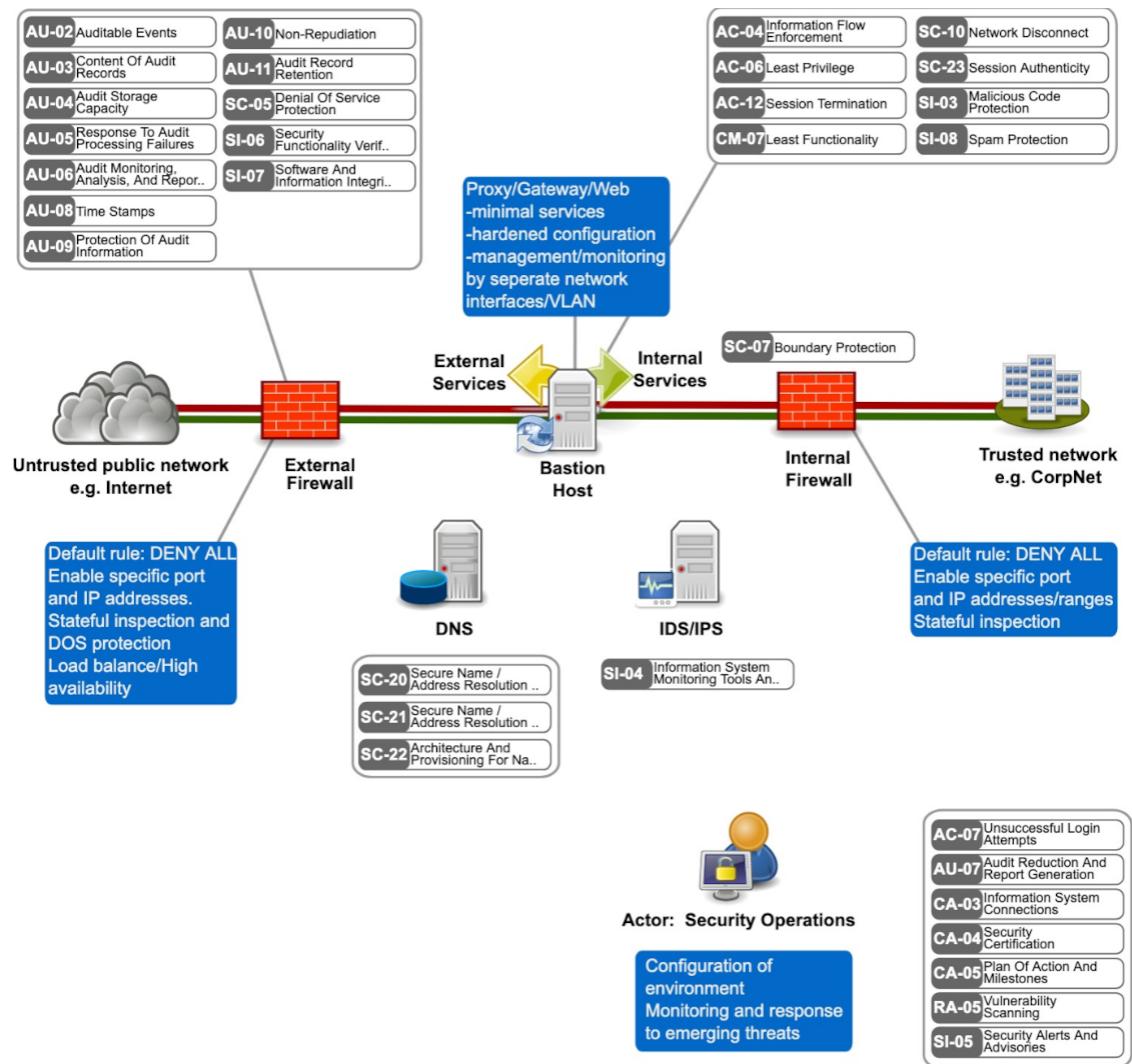
- Least privilege: Jede Funktion mit minimalen Rechten ausstatten -> Jeden Benutzer die minimal nötigen Rechte zuweisen -> Reduzierung Auswirkung im Falle von Missbrauch / Kompromittierung

- Berechtigungen: Zugriff aus System, System-Berechtigungen (Lese/Schreibrechte auf Dateien), Anwendungsspezifische Rechte
- Prinzip: Alles verboten -> Rechte kontrollierte Öffnung
- Erfordert Authentifizierung

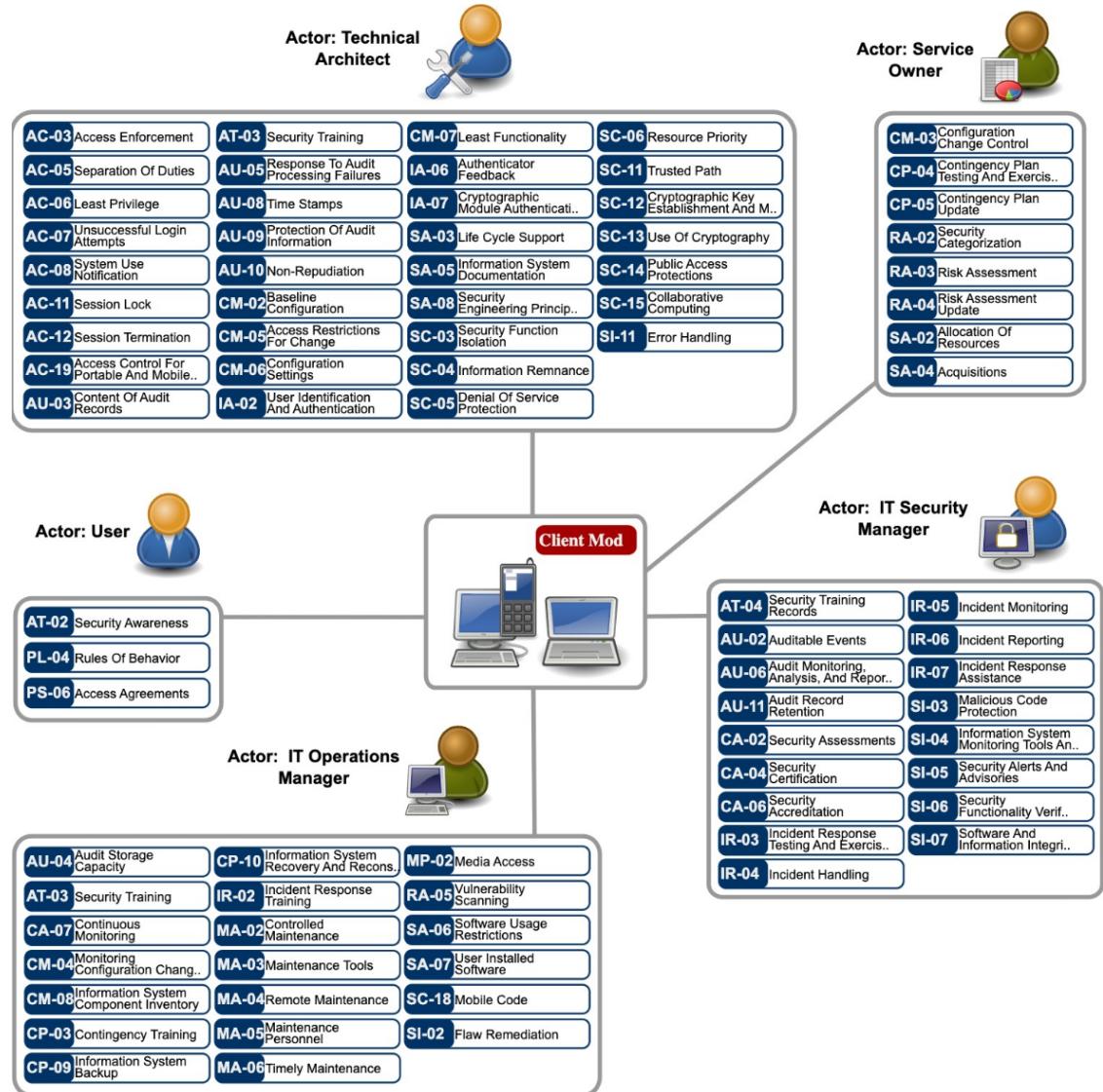
- Audit trails: Sichere, automatisiert erstellte und mit Zeitstempeln versehene Aufzeichnung von Ereignissen in einem System -> Nachverfolgung

- Manipulationsschutz v. Logdateien und Sammlung v. Logdatein an zentralem Ort (siehe SIEM) + Dokumentation v. Änderungen am System
- Security by Default:
 - Benutzer ändern die Vorkonfiguration selten, IoT oft nicht konfigurierbar
 - Daher: Konfiguration Standardzustand maximal sicher notwendig + Änderungen ermöglichen
- Failing Securely:
 - Fehlzustände werden oft ausgenutzt -> Trotzdem Sicherheitsziele d. Systems eingehalten werden -> Rückabwicklung von Änderungen nach einem Fehler -> Zur Not: Sicherer Default-Fall ("Secure Default")
- Security Architecture Patterns:
 - Security Controls:
 - Technical:
 - AC: Access Control
 - AU: Audit And Accountability
 - IA: Identification And Authentication
 - SC: System And Communications Protection
 - Management:
 - CA: Certification, Accreditation And Security Assessments
 - RA: Risk Assessment
 - SA: System And Services Acquisition
 - Operational:
 - AT: Awareness And Training
 - CM: Configuration Management
 - CP: Contingency Planning
 - IR: Incident Response
 - MA: Maintenance
 - MP: Media Protection
 - PE: Physical And Environmental Protection
 - PL: Planning
 - PS: Personnel Security
 - SI: System And Information Integrity

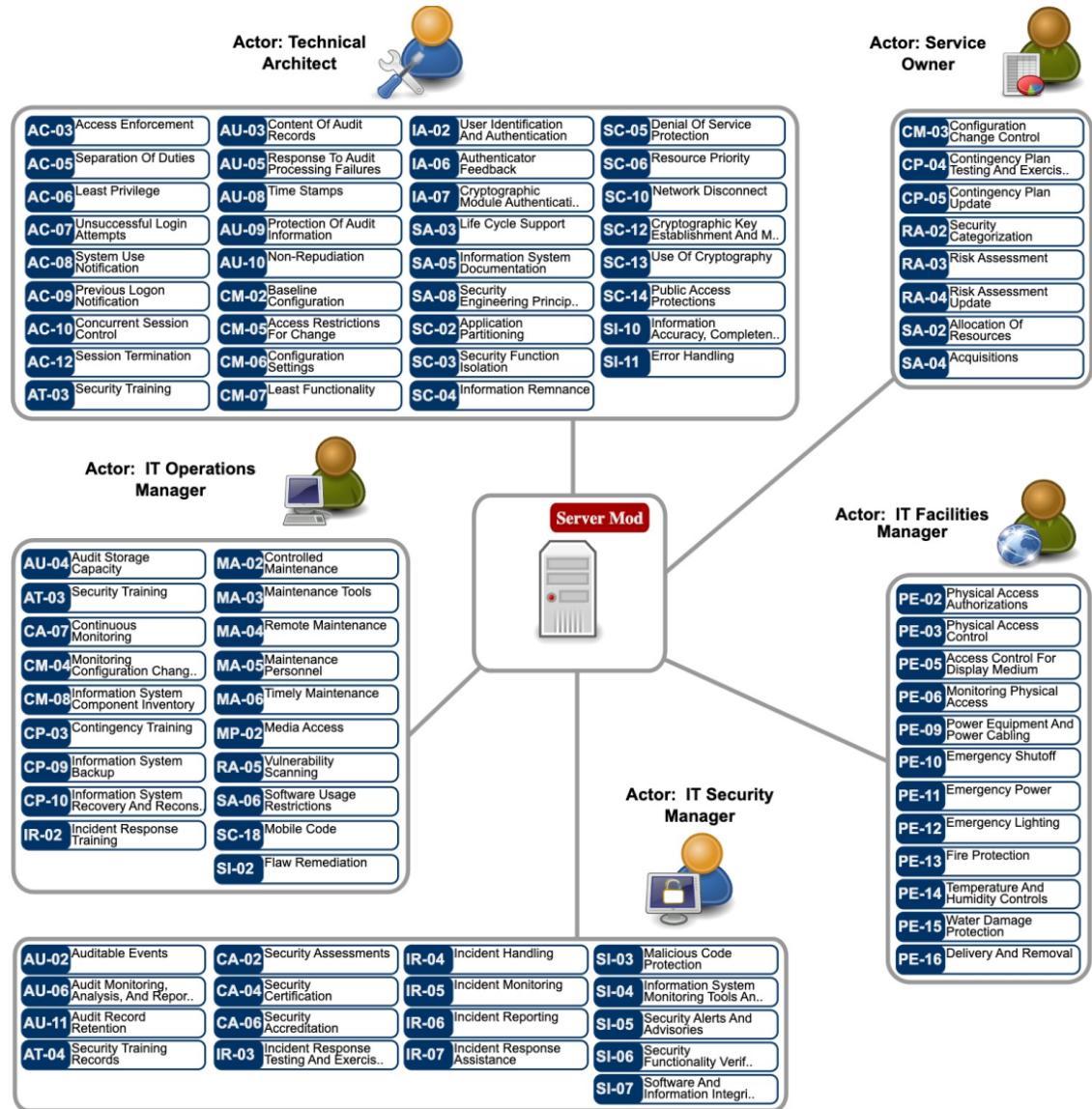
■ DMZ:



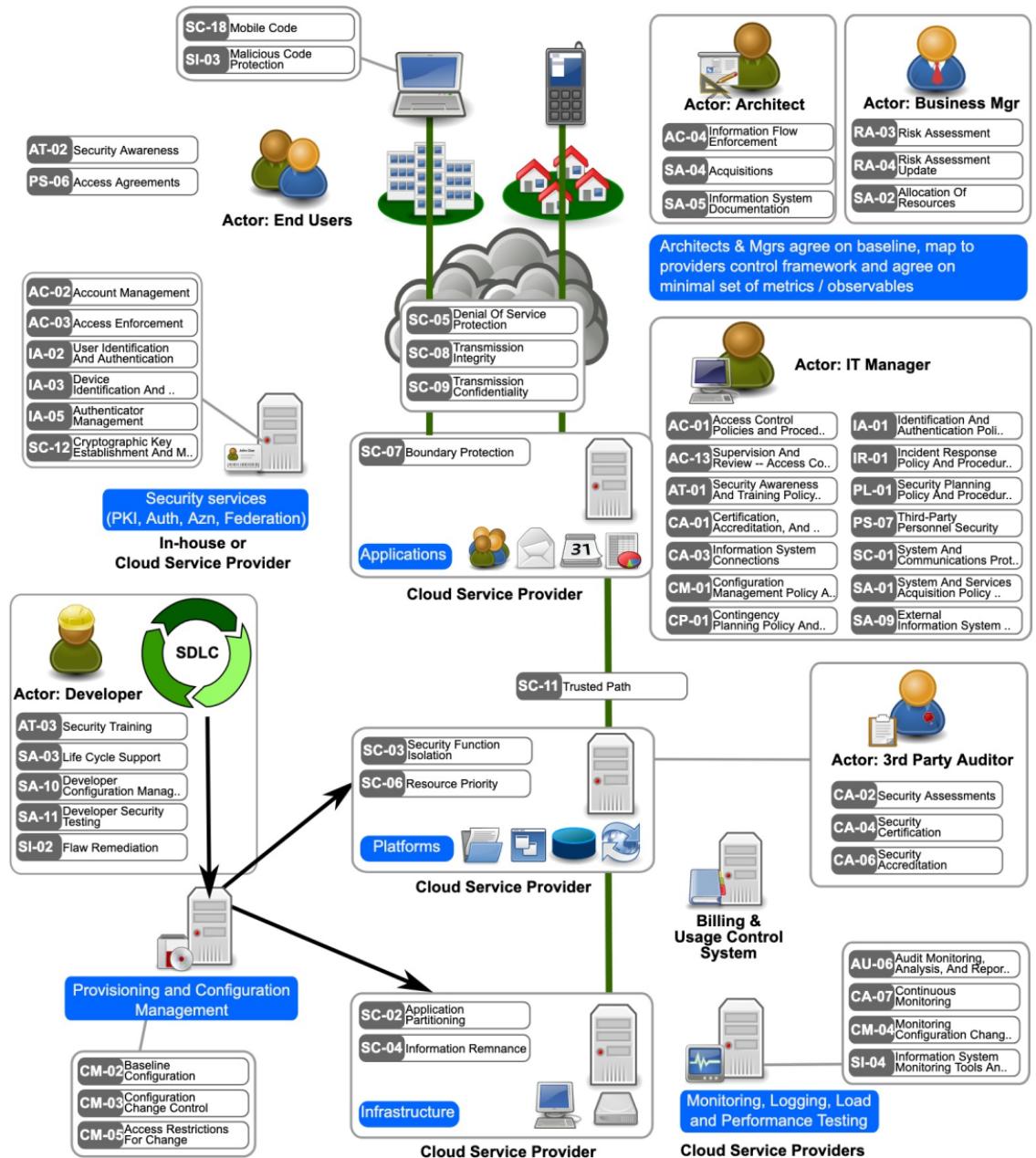
- Client:



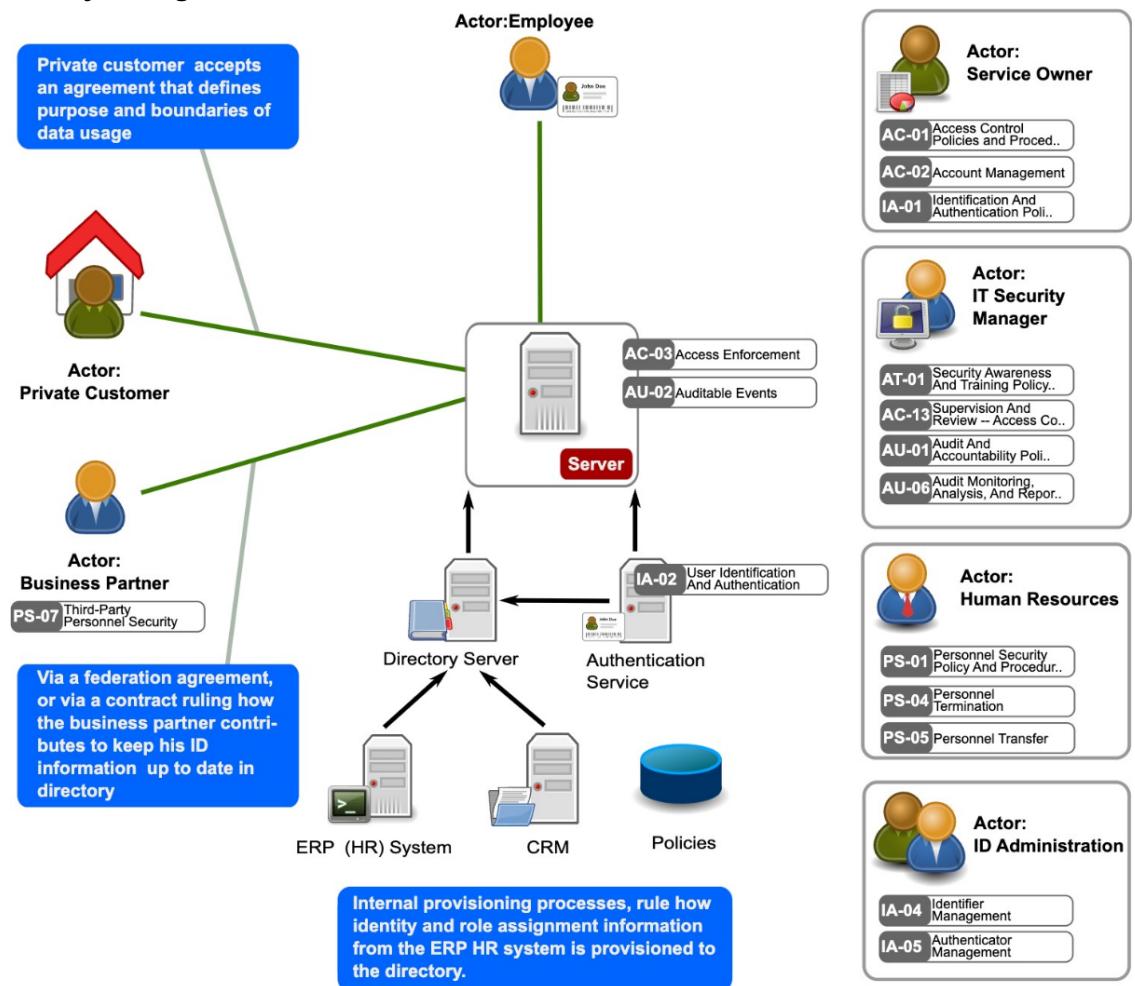
■ Server:



■ Cloud:



- Identiy Management:



5. Modellierung von IT-Sicherheit im Security Engineering

- Modell: Abbildung / Vereinfachung / Entwurf der Wirklichkeit um komplexe Aspekte verständlicher zu beschreiben

Angreifermodellierung

- Annahmen über Angreifer -> Dient Grundlage d. Systementwurfs (Explizit ausdrücken und festhalten)
- Klassifikation:
 - Quantitativ: Einschätzungen Wertbereichs in vorgegebenen Kategorie
 - Qualitativ: Qualitative Annahmen über Eigenschaften eines Angreifers
- Dolev-Yao Angreifermodell:
 - Analyse von Kryptographischen Protokollen
 - Angreifer ist ein aktiver, omnipotenter Saboteur (Kommunikation im Netzwerk abfangen, abhören, modifizieren jedoch Kryptografie nicht berechnen)
 - Ziel: Formaler Beweis über die Sicherheit v. Protokollen
 - Angreifer sehr stark modelliert
- Ponikwar-Hof-Modell:
 - Realistischere Abbildung der Eigenschaften von Angreifern basierend auf deren Ziel
 - Nutzbarkeit für Szenarien
 - Qualitatives Modell
 - Spezialisiert auf Anwendungsszenarien Cooperative, Connected, Automated Mobility (CCAM)

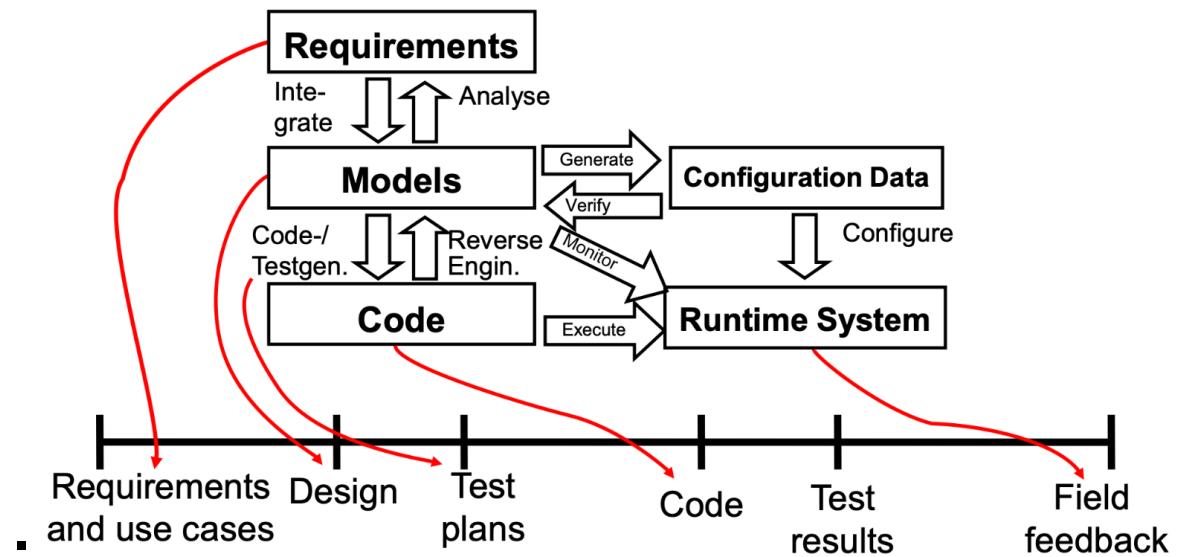
- Angreifermodellierung:
 - Zugang betrachteten Mobilitätssystem: Insider Attacker vs. Outsider Attacker
 - Aktivität: Active Attacker vs. Passiv Attacker
 - Anpassungsfähigkeit: Static Attacker vs. Dynamic Attacker
 - Kooperation: Cooperative Attacker vs. Individual Attacker
 - Zugang Mobilitätssystem: Local Attacker (Physischer Zugang) vs. Extended Attacker (1-Hop-Funkverbindung) vs. Global Attacker
 - Vorgehen: Malicious Attacker vs. Rational Attacker vs. Opportunistic Attacker

Modder / Tuner

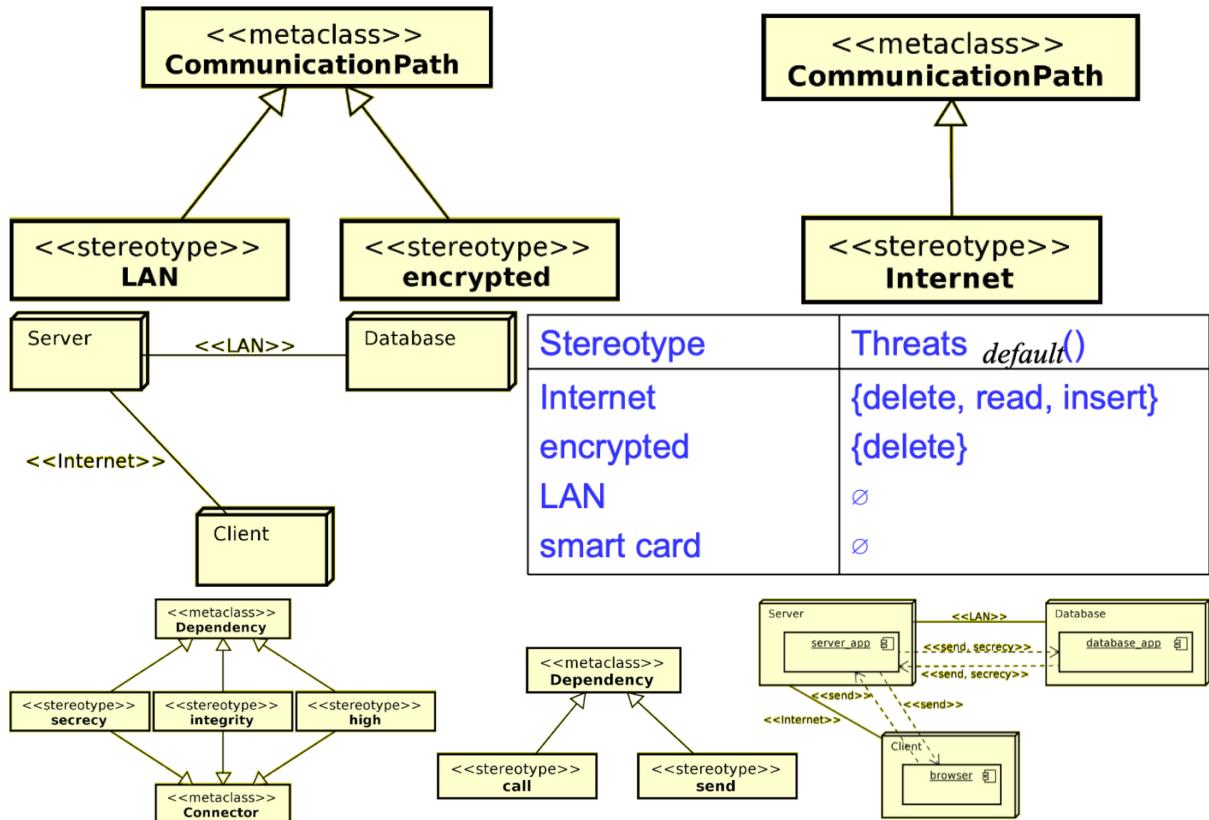
Attacker Properties			
Membership:	insider	outsider	
Method:	active	passive	
Adaptability:	dynamic	static	
Organization:	cooperative	individual	
Scope:	global	extended	local
Method:	malicious	rational	opportunistic

UML Security (UMLsec)

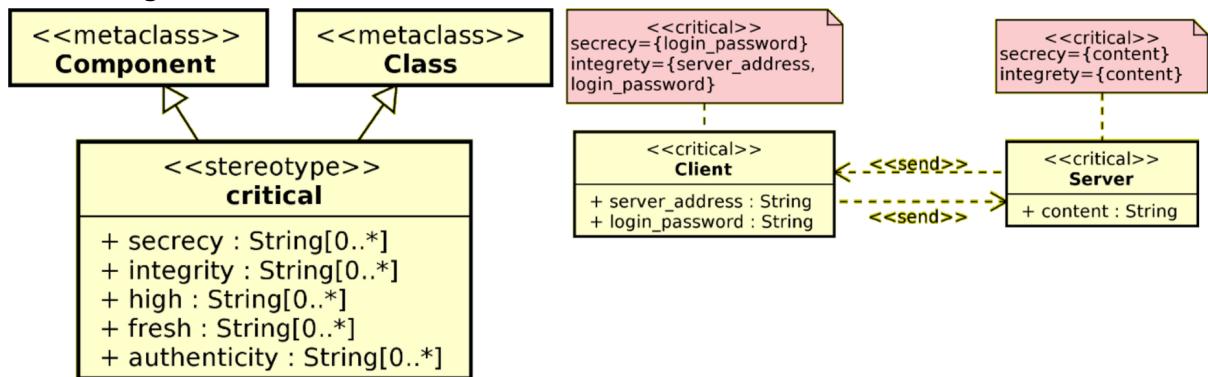
- Ziel: Sicherheitsanforderung in UML sichtbar machen (Security by Design) + Formale Verifizierung d. erzeugten Modells auf Sicherheitseigenschaften
- Misuse Case Diagramm:
 - Bedrohung in UML früh sichtbar
 - Mitigations-Maßnahmen früh sichtbar
 - Qualitatives Modell
 - Idealerweise für Misuse Cases mit hohem/kritischen Risiko (aus Bedrohungs- u. Risikoanalyse)
 - Mitigation-Use Cases werden in weiteren Betrachtung mitgeführt
- UMLsec:
 - Qualitatives Modell
 - Ergänzt um Security-Annotationen (Tags)
 - Vorgefertigte Tags für wiederkehrende Sicherheitsanforderungen und Angriffsszenarien
 - Kann formale Verifikation der Einhaltung dieser Security-Annotationen münden
 - Ziel:
 - Entwickler soll können:
 - Korrektheit eines Security Designs überprüfen
 - Security-Mechanismen korrekt in einen Systemkontext einbetten
 - Größere Systembestandteile auf deren Security hin analysieren



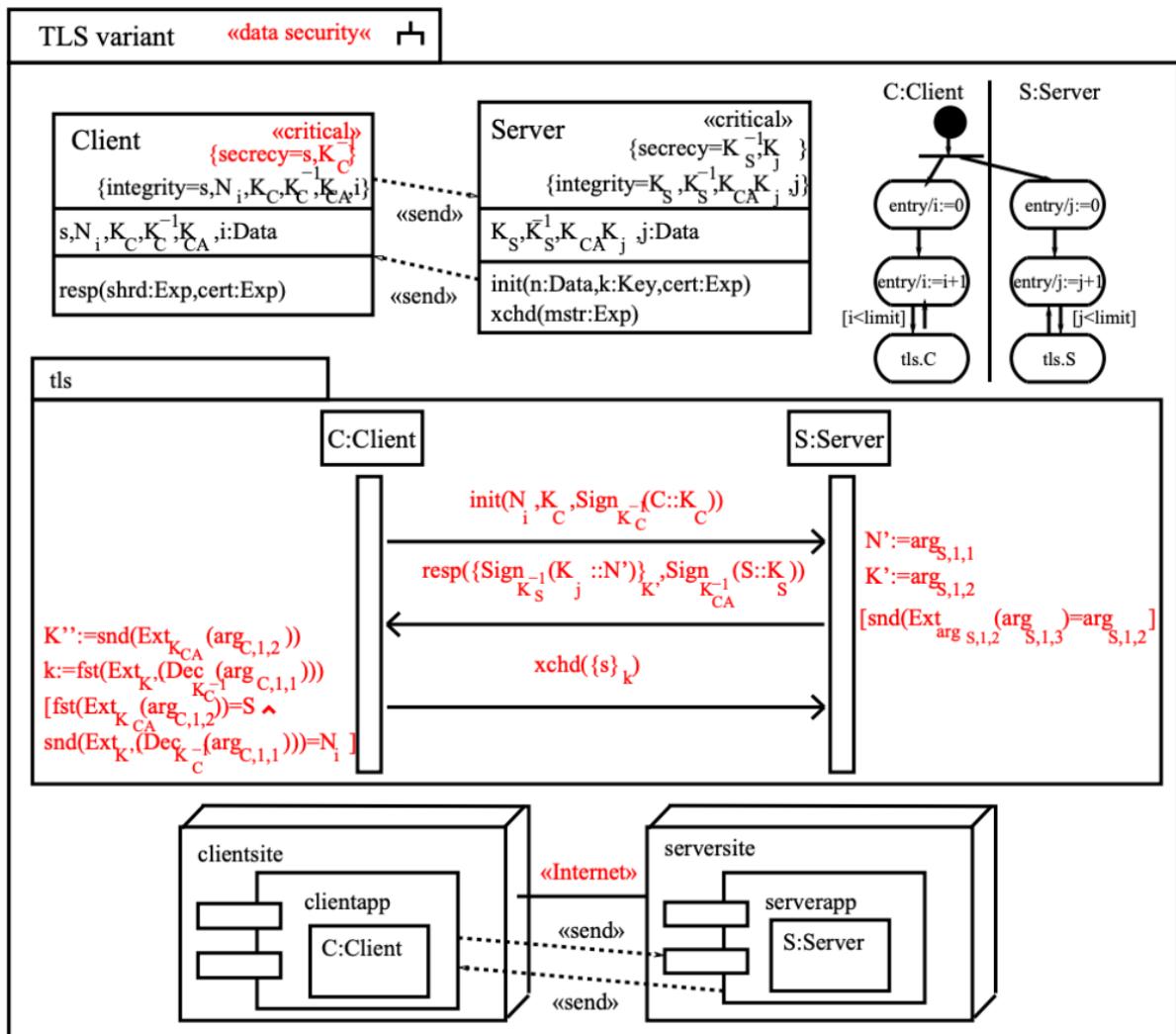
- Verwendet Stereotypen (⟨⟨...⟩⟩) Modellierung von Sicherheitseigenschaften
- Ergänzen UML Metamodell
- Ergänzt:
 - Aktivitätsdiagramm: Sicherer Kontrollfluss
 - Klassendiagramm: Datenaustausch entsprechendes Sicherheitsniveau
 - Sequenzdiagramm: Sicherheitskritische Interaktion
 - Zustandsdiagramm: Sicherheitseigenschaften innerhalb Klasse
 - Verteilungsdiagramm: Sicherheitsanforderungen physikalische Komponenten
 - Paketdiagramm: Ganzheitliche Sicht auf Security
- Verteilungsdiagramm:



- **Klassendiagramm:**



- Vereinigung:



6. Security Testing

- Testen: Prozess, um Zustand eines Systems/einer Anwendung gegen Menge von Kriterien zu vergleichen
 - Kosteneinsparung: Testen im gesamten Software Development Life Cycle (SDLC)
 - Beispiele in Phasen:
 - Anforderungsphase: Normative Vorgaben in Bezug auf IT Sicherheit eingehalten?
 - Entwurfsphase: Sicherheitsanforderungen mit Entwurf prinzipiell umsetzbar?
 - Implementierungsphase: Erfüllung von nicht-funktionalen und funktionalen Sicherheits-Anforderungen testen

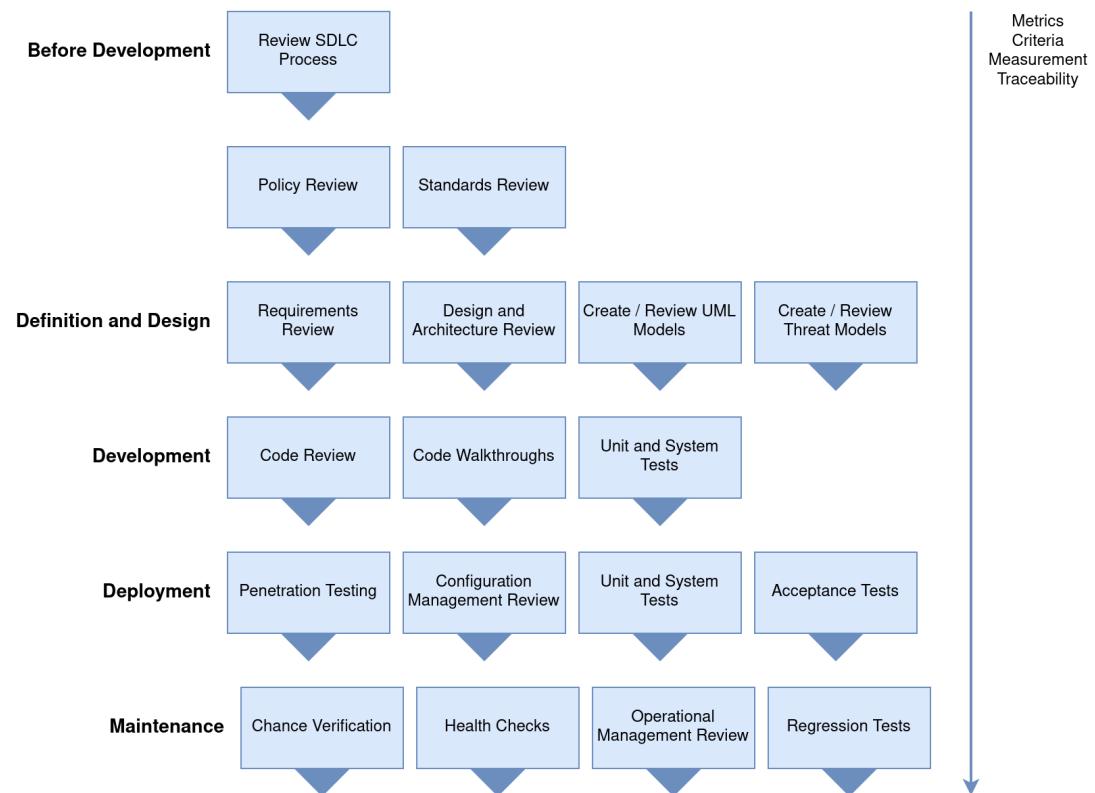
- Ausbringen: Überprüfung Konfigurationsoptionen vor Auslieferung, Tests von ausgewählten Installationen im Betrieb
 - Wartung: Bei Regressions- und Integrationstest von Patches prüfen, ob neue Sicherheit sichergestellt ist
 - Sichere Software erfordert Betrachtung aller Faktoren: Mensch (Ausbildung, Awareness), Prozesse (Prozesse, Regeln, Durchsetzung), Technik (Prozess in Implementierung umgesetzt)
 - Prinzipien:
 - Weitreichend denken: Von einzelnen Verwundbarkeiten lösen sondern Teile des Systems identifizieren die das größte Risiko haben (Besser für Verwundbarkeits Fenster)
 - Es gibt keine Allzwecklösung
 - Testen am besten im Rahmen eines SDLC -> Kosten
 - Umfang der Sicherheit muss verstanden werden
 - Früh testen und oft testen
 - Was ist notwendiges Sicherheitslevel?
 - Sicherheit denken lernen ("outside of the box")
 - Annahmen hinterfragen und außerhalb vom normalen testen
 - Verständnis für Testsubjekt (Dokumentation)
 - Falsches Gefühl der Sicherheit vermeiden
 - Wo möglich Source Code testen
 - Entwicklung von Metriken -> Wird etwas besser?
 - Dokumentation der Ergebnisse der Tests
 - Unterschied zu normalen Softwaretests
 - Nicht-Sicherheits-Softwaretests:
 - Tester sucht „positive“ Ergebnisse
 - Anforderung in Software umgesetzt
 - Nichterfüllung einer Anforderung wird als Feststellung aufgenommen
 - Testet überwiegend funktionale Anforderungen
 - Sicherheits-Softwaretests:
 - Gezielte Suche nach „negativen“ Ergebnissen
 - Nichterfüllung einer (Sicherheits-) Anforderung festgestellt => „weiter bohren“
 - Weiterhin: Grad der Anfälligkeit für Attacken zu identifizieren
 - Getestet werden überwiegend nicht-funktionale Eigenschaften
 - Sicherheits-Softwaretest != Test auf funktionale Sicherheit
 - Sicherheits-Tests zusammen mit normalen Tests durchführen
 - Vorgehen (ähnlich normalen Tests)
 - Testfälle definieren, Dokumentation Testfälle, Testplan
 - Nicht nur Testfälle nur aus Anforderungsdefinition sondern zusätzlich Gesetzliche Anforderung, Standards, interne Regelungen, Angriffsvektoren aus Bedrohungsmodellierung, Aktuelle Angriffe auf ähn. Software, Verwendete Sicherheitsentwurfsmuster
- Methodik:
 - Penetrationstest:
 - Unstrukturierten Herangehensweise (Menschliche Intuition und Erfahrung)
 - Strukturiertes Vorgehen
 - Toolbasierte automatisierte Überprüfung
 - Ggf. KI

- 2 Arten (Ziel gleich):
 - Blackbox:
 - Hacker hat initial kein Wissen über Zielsystem
 - Testumfeld gleicht Bedingungen für echten Hacker
 - Vorgehen: Aufklärung mittels automatisierter Tools -> Identifizierte Schwachstellen genauer analysieren und gezielt ausnutzen (Informationsbeschaffung) -> Angriffe auf "übliche" Stellen (Zeitliches Limit) -> Fuzzing
 - Pro: Schnell umzusetzen, umfangreiches Ergebnis bei geringem Einsatz von Mitteln, kann auch von Externen durchgeführt werden, testet Angriffsfläche nach außen
 - Con: Testet nur Angriffsfläche nach außen, nicht komplett
 - Whitebox:
 - Hacker hat Zugang zum Sourcecode
 - kombinierbar mit klassischen strukturbasierten Testverfahren
 - Dokumentation prüfen u. Erstellen v. Angriffsvektoren -> Angriffsplanung (Dokumentation + Sourcecode) -> Durchführung Angriff + protokollieren
 - Pro: Deutlich zielgerichteter und genauer als Blackbox-Penetrationstests
 - Con: Aufwendig, lange Vorbereitungszeit
 - Im Vergleich zu Code Review: Schneller, Weniger Kompetenz nötig, Testet ausgebrachten Code -> Zu spät in SDLC, Überprüft nur Angriffe d. ersten Verteidigungslinie
 - Whitebox kosteneffizient > Blackbox
 - Tiefes Eindringen wenig kosteneffizient (Vulnerability Assessment)
 - Beheben von Verwundbarkeite aufwendiger als Finden
- Code Review:
 - Ergebnis Maßnahmenpaket zur sicheren Softwareentwicklung
 - Benötigt vollständige Dokumentation zusätzlich zum Code
 - Komplexitätsstufen:
 - Informelles Review: Keineformalen Anforderungen
 - Geführtes Review: Entwickler führt Gruppe v. Experten durch Code
 - Technisches Review: Peer-Entwickler aus anderen Gruppen und externe Experten identifizieren Schwachstellen
 - Inspektion: Professioneller Moderator führt kompletten Code-Review-Prozess unter Beteiligung betroffener Entwickler, Peer-Entwickler und externen Experten, Maßnahmen zur Behebung werden sofort beschlossen
 - Pro: Gründliche Prüfung des Vorgehens
 - Con: Evtl. teuer und langwierig, benötigt Experten, Laufzeitfehler nicht erkennbar
- Am meisten getestet in Define, Design, Develop

OWASP

- WSTP (Web Security Testing) Guide mit Testtechniken, OWASP Testing Framework, Konkrete Techniken, Reporting
- OWASP Testing Framework:
 - Standardfragen: Vollständig?, Angemessen?, Berücksichtigt Regulatorien und Standards?, Realistisch?, Kann so etwas bestimmtes erfüllt werden?, Auf dem Stand der Technik?

- 5 Phasen:
 - Before Development:
 - Schwerpunkt auf der Betrachtung von Prozessen und Regeln
 - Überprüfung SDLC (Vollständig / Aktuell)
 - Überprüfung von Regeln und Standards (Vollständig / Aktuelle Bedrohungen / Akteulles Regulatorien)
 - Überprüfungen Metriken und Definition von Metriken für die Nachverfolgung
 - Definition and Design:
 - Schwerpunkt auf der Überprüfung von Design Artefakten
 - Überprüfung der Anforderungen
 - Überprüfung Bedrohungsmodell
 - Überprüfung Design und Architektur
 - UML Modelle (Ergänzung UMLsec / Erstellung Modelle für security-kritische Use Cases)
 - Development:
 - Schwerpunkt auf der Überprüfung des erzeugten Codes
 - Code Review, Code Walkthroughs, Unit and System Tests
 - Deployment:
 - Schwerpunkt auf Überprüfung der ausgebrachten Binaries
 - Überprüfung Konfigurationsmanagement
 - Penetration Testing
 - Maintenance:
 - Schwerpunkt auf Aufrechterhaltung des Sicherheitsniveaus
 - Regelmäßige Überprüfung des operativen Betriebs von Anwendung und zugehöriger Infrastruktur
 - Monatlich/Quartalsweise Überprüfung auf neue Security Risiken und auf angemessenes Sicherheitsniveau
 - Für jede Änderung am System (z.B. Patch, Konfigurationsänderung) muss sichergestellt werden, dass Security-Tests durchgeführt werden



- Code Review:
 - Entwickler ggf. System-Architekt führen Security Team durch Code -> Verständnis für Testgegenstand und Motivation für gewählte Lösung
 - Anschließend Suche nach Sicherheitslücken / Verletzung von Regeln, Standards, Richtlinien
 - OWASP Code Review Guide:
 - Risiko-basierter Ansatz:
 - Review-Intensität eines Moduls hängt vom assoziierten Risiko ab.
 - Risiko ermitteln (Ease of exposure, Value of loss, Regulatory controls) -> Behandlung (Reduce, Accept, Avoid)
 - Wichtig für Risikoabschätzung: Überblick über Anwendung, Kontext (Notwendiges Sicherheitsniveau / Einsatzzweck), Sensible Daten, Benutzerrollen und Zugriffsrechte, Verwendete Programmiersprachen und deren Security-Eigenschaften, Standards und Richtlinien des Unternehmens
 - Checkliste für Schwachstellenbereiche:
 - Datenvalidierung
 - Authentifizierung
 - Session Management
 - Autorisierung
 - Kryptographie
 - Fehlerbehandlung
 - Logging
 - Security Konfiguration
 - Netzwerkarchitektur
 - Bedrohungsmodellierung:
 - Zielt ebenfalls darauf hin, risikobasiertes Code Review durchzuführen
 - Geeignet wenn Security Team nicht in SDLC eingebunden

- 3 Schritte: Decompose the Application, Determine and rank Threats, Determine Countermeasures and Mitigation
 - zu Decompose: Externe Abhangigkeiten identifizieren, Entry Points (= Attack Vectors) identifizieren, Werte identifizieren, Angriffsoberfache identifizieren, Zugriffsrechte fur externe Entitaten, Datenflussanalyse, Transaktionsanalyse