Einführung in Security

Einführung in die IT Sicherheit

Definition Sicherheit

Sicherheit ist eine Sachlage, in der das **Restrisiko nicht größer als das Grenzrisiko** ist

Das **Grenzrisiko** ist das größte noch vertretbare Risiko eines bestimmten technischen Vorganges oder Zustandes.

Eine absolute Sicherheit ohne jegliches Risiko gibt es weder in der Technik noch in der Natur

Definition Risiko

Risiko = Schaden * Eintrittswahrscheinlichkeit

Sicherheitsziele/Schutzziele

CIA-Triad

- **Confidentiality (Vertraulichkeit):** Informationen können nur durch Berechtigte eingesehen werden
- **Integrity:** Informationen werden so wiedergefunden, wie sie versendet wurden - keine unautorisierte Veränderung
- **Availability:** Berechtigte Können in einer definierten Zeit auf Informationen zugreifen

Weiter Sicherheitsziele

- **Authentizität:** Nachlesen, von wem eine Information gespeichert/gelesen/geändert?
- **Nichtabstreitbarkeit:** Kann nicht abstreiten, von wem eine Information gespeichert/gelesen/geändert

Weiter Definitionen

- **Bedrohung/Threat:** Potentielle Verletzung der IT Sicherheit
- **Angriff/Attack:** Aktion, die eine Bedrohung wahr werden lässt
- **Schwachstelle/Vulnerability:** Sicherheitsfehler im System, welchen man für Angriff ausnutzen kann

- **Exploit:** Software, die eine Schwachstelle ausnützt

Kategorien für Angriffe

1. **Unberechtigter Zugriff auf Daten:** Sniffing, MiM (man in the middle)
2. **Verbreitung/Akzeptanz falscher Informationen:** Spoofing, MiM
3. **Unterbrechung der Funktionalität:** (D)DOS
4. **Widerrechtliche Verwendung:** Command Injection

Sniffing

Abfangen/Mitlesen von Datenpaketen in Netzwerken - meist zur Analyse/Abhören sensibler Informationen

Spoofing

Vortäuschen einer Identität, um sich Zugriff auf ein System/Daten zu verschaffen, indem man die Quelle von Kommunikation & Daten fälscht

Phasen eines Angriffs

- Sammeln von Daten über das Angriffsziel
- Ausnutzen von gefundenen Sicherheitsproblemen (SQL Injection)
- Aufrechterhaltung des Zugriffs (Benutzerkonto anlegen)
- Verwischen von Spuren (löschen der Logs)

Risiko- & Bedrohungsanalyse

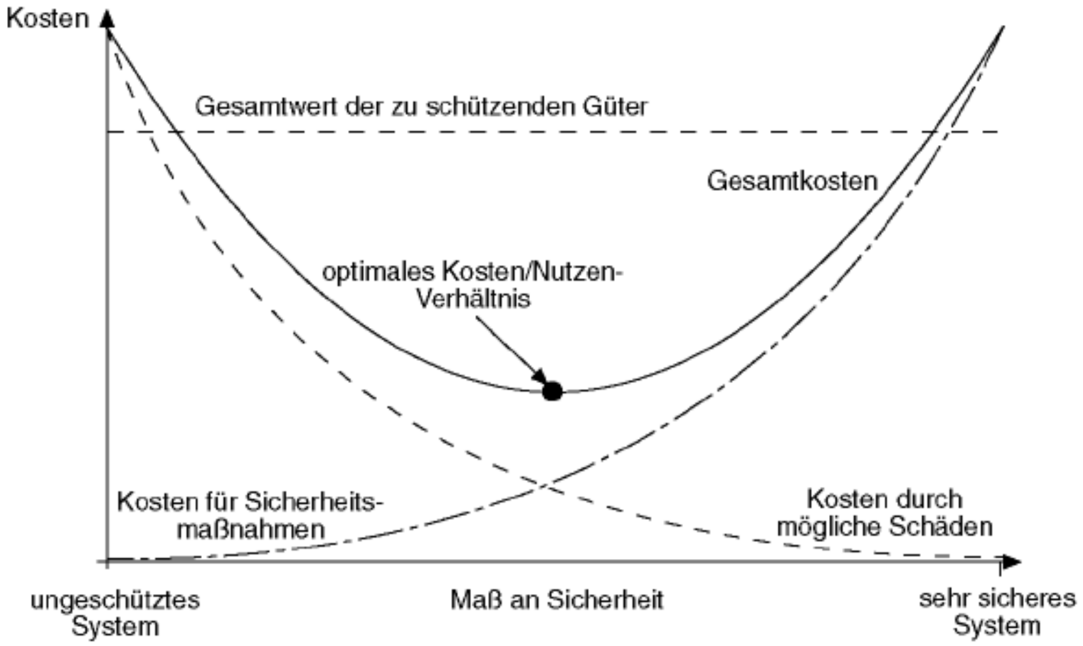
- Gruppieren der Bedrohung (z.B. nach Zielobjekt, Urheber:in, Motivation, Absicht)
- Auflistung der Bedrohungen
- Risikoanalyse/-bewertung
- Sicherheitsmaßnahmen
- Restrisikoabschätzung

Bedrohungspotential von Angreifenden

- Skill Level & Know How (Script Kiddies, White Hat, etc)

- Budget
- Zeit

Kosten- Nutzenanalyse für IT-Sicherheit



Lösungsansätze

Security, Architektur, Designprinzipien

- Security von Beginn an beim Design berücksichtigen
- Genau definierte Aufgaben und Interfaces
- Verwendung von Standards

Kryptographie

Definiton

Wissenschaft zur Geheimhaltung von Nachrichten

Unverschlüsselter Text *(Plaintext)* wird mittels einer Funktion, die durch einen Schlüssel *(Key)* parametrisiert wird in verschlüsselten Text *(Ciphertext)* umgewandelt.

Einfache Berechnung eines Ergebnisses, schwierig die Berechnung umzukehren.

Kerckhoffs' Prinzip

> Sicherheit darf nur von Geheimhaltung des Schlüssels abhängen. Nicht von der Geheimhaltung des Algorithmus!

Hash-Verfahren

- Ein Hash ist ein eindeutiger Fingerprint eines Dokuments
- **Kollisionsresistenz:** Zwei Dokumente mit unterschiedlichem Inhalt dürfen nicht den gleichen Hash haben
- **Einwegfunktionen:**
  - y= f(x) ist mit wenig Aufwand zu berechnen
  - Umkehrfunktion x = f−1(y) nicht/schwer anwendbar
- **Beispiele:**
  - MD5, SHA-1 → unsicher
  - SHA-2 Familie (SHA-224, SHA-256, SHA-384 und SHA-512)
  - SHA-3 Keccak Algorithmus wurde von NIST 2012 ausgewählt

Symmetrische Kryptographie

- Man verwendet zum Ver- und Entschlüsseln den selben Key als Parameter

Beispiele

- Der **Advanced Encryption Standard (AES)** wurde im Jahr 2000 als Nachfolger des Data Encryption Standards (DES) eingeführt und gilt als pragmatisch sicher. Es verwendet den **Rijndael**-Algorithmus.

Nachteile:

- Schlüsseltausch bei zu vielen Teilnehmenden
- wenn jemand ausgeschlossen werden muss, muss der Schlüssel von allen geändert werden

Asymmetrische Kryptographie

Asymmetrische Verschlüsselung

1. Sender verschlüsselt Nachricht mit Public Key des Empfängers
2. Empfänger entschlüsselt Nachricht mit seinem Private Key

Asymmetrische Signatur

1. Sender erstellt eine **Hash-Wert** (z. B. SHA-256) der Nachricht, verschlüsselt ihn mit seinem Private Key und hängt die Signatur an die Nachricht an.
2. Empfänger entschlüsselt die angehängt Signatur mit dem Public Key des Senders und vergleicht den Hash-Wert mit dem berechneten Hash-Wert.

Beispiele

- Rivest-Shamir-Adleman (RSA)
- Optimal Asymmetric Encryption Padding (OAEP)

Nachteile

- Rechenaufwändig

Hybridverschlüsselung

- Symmetrische Verschlüsselung: Nachteil Schlüsselaustausch
- Asymmetrische: Rechenaufwändig
- Lösung: Kombination

Ablauf

- Verschlüsselung der Daten mit neu erstelltem symmetrischen Schlüssel
- Verschlüsselung des symmetrischen Schlüssels mit Public Key
- Asymmetrisch verschlüsselten symmetrischen Schlüssel an das symmetrisch verschlüsselte Dokument anhängen

Probleme Kryptographie

- **Komplexität** kryptographischer Verfahren und PKIs erhöht die Anzahl an potenziellen Fehlern.

- **Alterung** von Algorithmen und Schlüssellängen, d.h. Angreifer können eventuell Daten ohne Kenntnis des privaten Schlüssels entschlüsseln (z.B. mittels Brute-Force)
- **Kompomitierung** oder **Verlust** des privaten Schlüssels
- **Integrität/Authentizität** für Schlüssel (z.B. Zertifikate enthalten eine digitale Signatur, um vor MitM-Attacken zu schützen)
- **Anonymität** von Schlüsseln

Chipkarten

Speicherkarten

Keine Sicherheitsmerkmale für Zugriffskontrolle vorhanden

Prozessorkarten

- Betriebssystem der Karte regelt Zugriff auf Daten
- Möglichkeit, Zugriff über PIN zu regeln
- Idealerweise verlässt Schlüssel nie die Karte

Beispiele

- Kontaktbehaftet/Kontaktlos
- Gesundheitskarte, SIM, Bankomatkarte

Angriffe

Side Channel Angriff:

- Versucht basierend auf der Implementierung eines kryptographischen Algorithmus' mit Informationen, die man beim Ablauf des kryptographischen Algorithmus' erhält, Informationen zum verwendeten Schlüssel zu finden.
- z.B. Stromverbrauch, Rechenzeit

Physikalisch

- mit extremen Temperaturen/Spannung Karte zu verhalten bewegen, das Angriff ermöglicht

Man in the middle

- Kartenterminal so manipulieren, dass andere Daten geschickt werden

Social Engineering

- Diebstahl
- Ausspionieren

Kartenterminals

Sicherheitsklassen

1. Keine besonderen Sicherheitsmerkmale
2. Gerät enthält PIN-Pad
3. PIN-Pad und Display
4. PIN-Pad, Display und eigenes Sicherheitsmodul zur Authentisierung des Geräts

Zertifikate

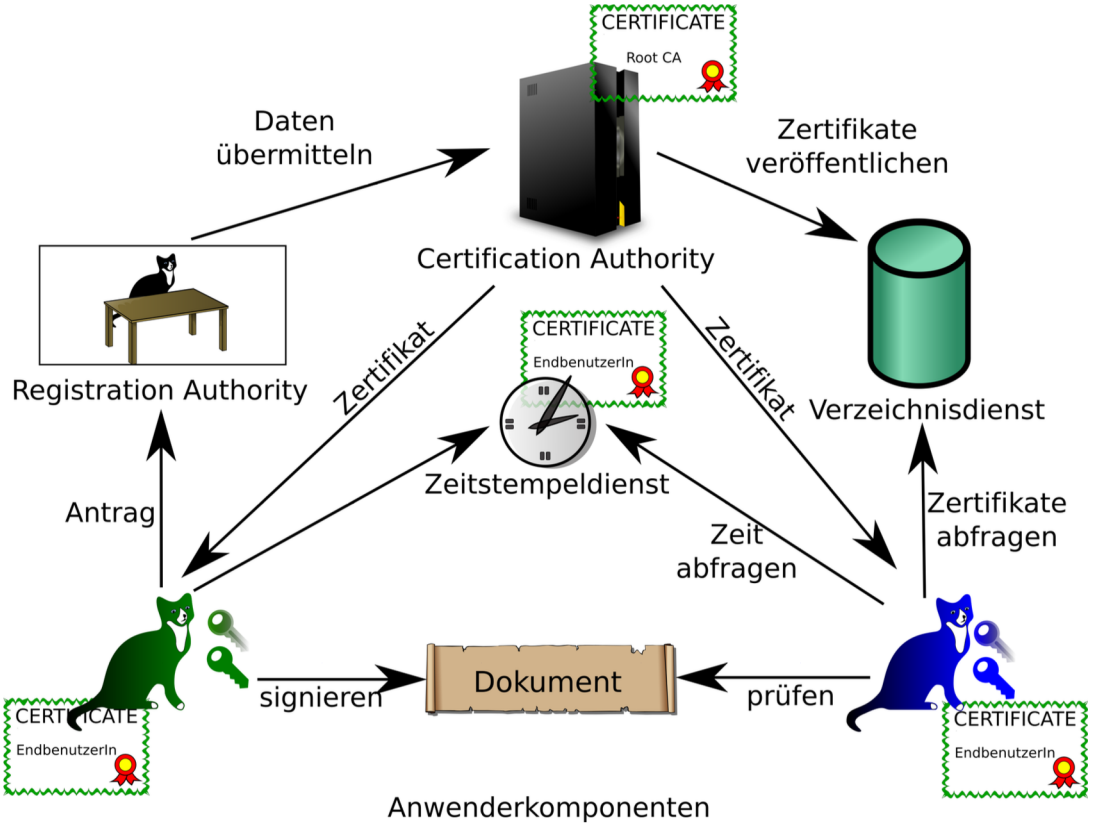
- Zuordenbarkeit eines Public Keys zu einer Identität
- Bestätigung durch CA
- im Rahmen der PKI öffentlich
- Zusätzliche Angaben (zeitliche Gültigkeit, Aussteller, Usage,..)
- **Beispiele:** X.509, CVC

PKI (Public Key Infrastructure)

Definition

Ein System unterschiedlicher Sicherheitskomponenten zur Erstellung, Verwaltung, Verteilung, Verwendung und Widerruf digitaler Zertifikate, um eine sichere Kommunikation und Authentifizierung zu ermöglichen

Bestandteile



- **Registration Authority (RA):** Prüft die Identität eines Antragstellenden & genehmigt/verweigert Ausstellung eines Zertifikats.
- **Certification Authority (CA):** Stellt Zertifikate aus, signiert sie mit privatem Schlüssel und verwaltet ihren Lebenszyklus. Bei Kompromittierung des privaten Schlüssels einer CA sind alle ausgeschriebenen Zertifikate und signierten Dokumente unsicher.
- **Verzeichnisdienst (DR):** Speichert Zertifikate und stellt sie sowie ihre Sperrinformationen
  (z. B. CRLs) öffentlich zur Verfügung. Jedes Zertifikat hat eine eindeutige ID zur Kennung des Besitzers.
- **Time Stamping Authority (TSA):** Stellt sicher, dass elektronische Dokumente oder Nachrichten mit einem vertrauenswürdigen Zeitstempel versehen werden, um ihre Integrität zu gewährleisten.
- **Sperrinformationen (CRL/OCSP):**
  - CRL (Certificate Revocation List): Liste gesperrter Zertifikate, die nicht mehr gültig sind

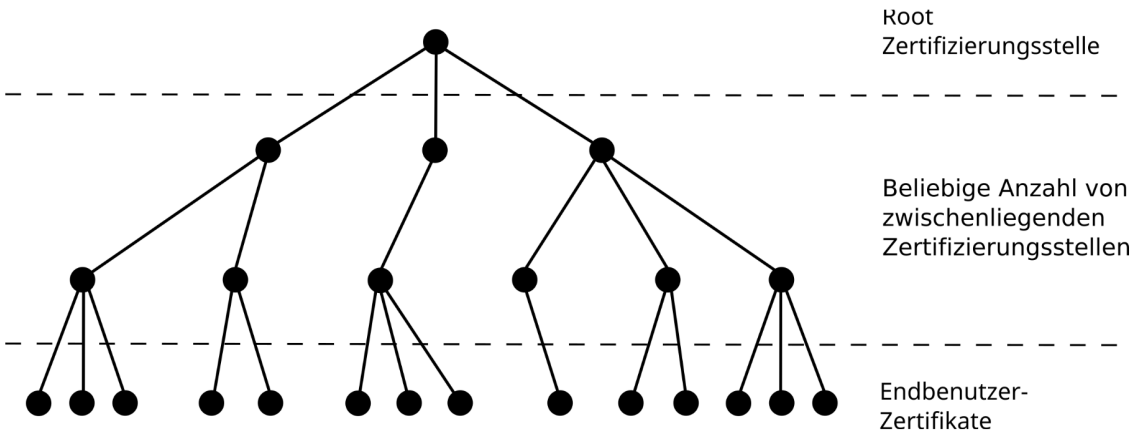
    - OCSP (Online Certificate Status Protocol): Ermöglicht die Echtzeitprüfung der Gültigkeit eines Zertifikats

- **Anwenderkomponenten:** Software und Hardware, die Zertifikate nutzen, z. B. Browser, E-Mail-Clients, oder Smartcards
- **Zertifikate:** Elektronische Dokumente, die einen Public Key mit der Identität des Besitzers verbinden. Sie sind von der CA signiert. Beispiel: X.509-Standard
- **Organisatorische Festlegungen:** Regeln und Richtlinien, die den Betrieb und die Nutzung der PKI bestimmen (z. B. Sicherheitsrichtlinien und Zertifikatsrichtlinien).

Vertrauensmodelle

Hierarchische Struktur

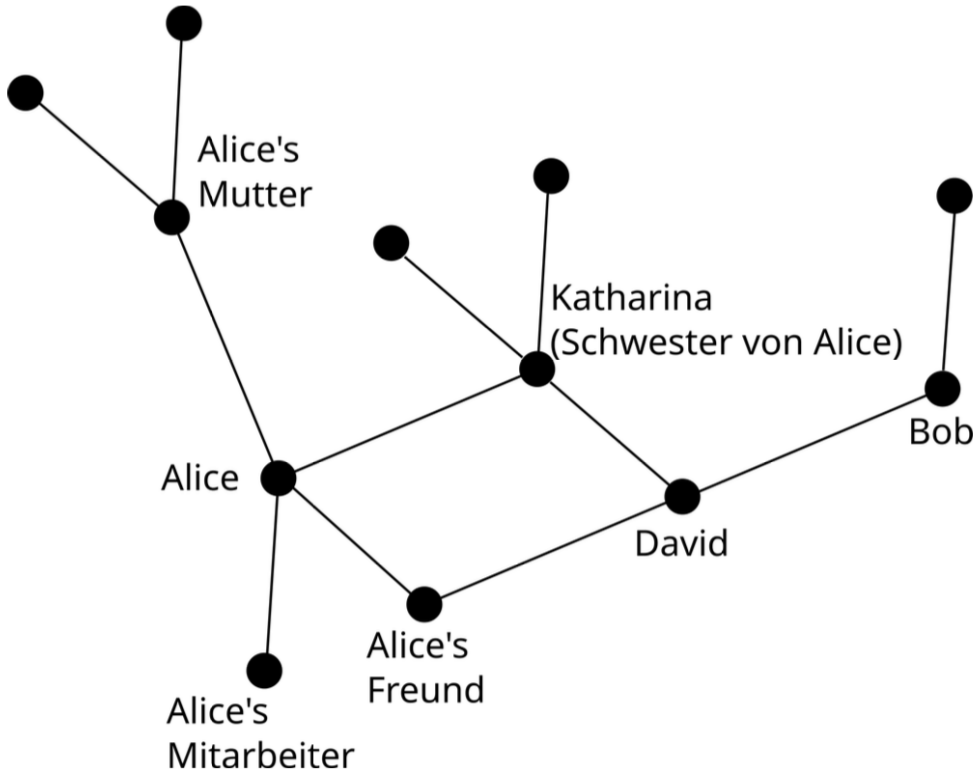
- Vertrauensstellung über Root CA
- Zwischenliegenden Zertifizierungsstellen wird indirekt vertraut
- Verwaltung großer Infrastrukturen möglich



Web of Trust

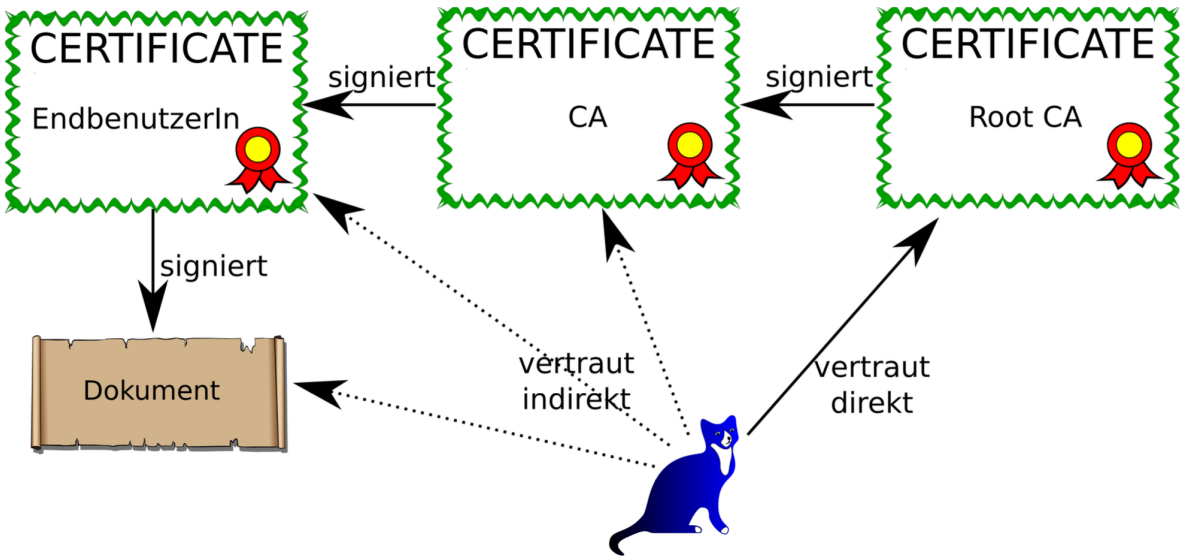
- Dezentral
- Direkte Vertrauensstellung, Beispiel: Pretty Good Privacy (PGP)
- Gegenseitige Beglaubigung von Nutzern, die ihre öffentlichen Schlüssel signieren (Keyrings), um Vertrauen dezentral aufzubauen, anstatt eine zentrale Autorität zu nutzen.

- Kommunikationsteilnehmende muss Zugehörigkeit des Schlüssels zur Identität der Kommunikationspartner:in überprüfen



Zertifikationskette

Als **Zertifikatskette** wird die gesamte Kette von der Root-CA über die Sub-CAs bis zum Endbenutzer:innen-Zertifikat verstanden.



Gültigkeitsmodelle

Gültigkeitsmodelle bei Signaturen legen die Regeln für die Überprüfung einer Signatur fest. Dabei wird die **gesamte Zertifikatskette** berücksichtigt.

Es gibt unterschiedliche Gültigkeitsmodelle:

- **Schalenmodell**: zum Zeitpunkt der Prüfung müssen alle Zertifikate im Pfad gültig sein.
- **Kettenmodell**: zum Zeitpunkt der Erstellung der Signatur muss jeweils das signierende Zertifikat gültig gewesen sein
- **Hybridmodell**: zum Zeitpunkt der Erstellung der zu prüfenden Signatur müssen alle Zertifikate im Pfad gültig gewesen sein

PCG/GPG

Pretty Good Privacy (PGP)

- Ursprünglich 1991 von Phil Zimmermann entwickelter Verschlüsselungsstandard
- **Kommerzielles Produkt,** das mehrmals den Eigentümer wechselte
- Basiert auf einem Public-Key-Verfahren mit öffentlichem und privatem Schlüssel
- **Verwendet hybride Verschlüsselung:** symmetrisch für Nachrichten, asymmetrisch für Schlüsselaustausch,

GNU Privacy Guard (GPG)

- Freie **Open-Source-Implementierung**, des OpenPGP-Standards

- Entwickelt als kostenlose Alternative zu PGP, veröffentlicht 1999
- Bietet ähnliche Funktionen wie PGP: Verschlüsselung und Signierung von E-Mails und Dateien
- Verfügbar für verschiedene Betriebssysteme wie Linux, MacOS und Windows

TLS (Transport Layer Security)

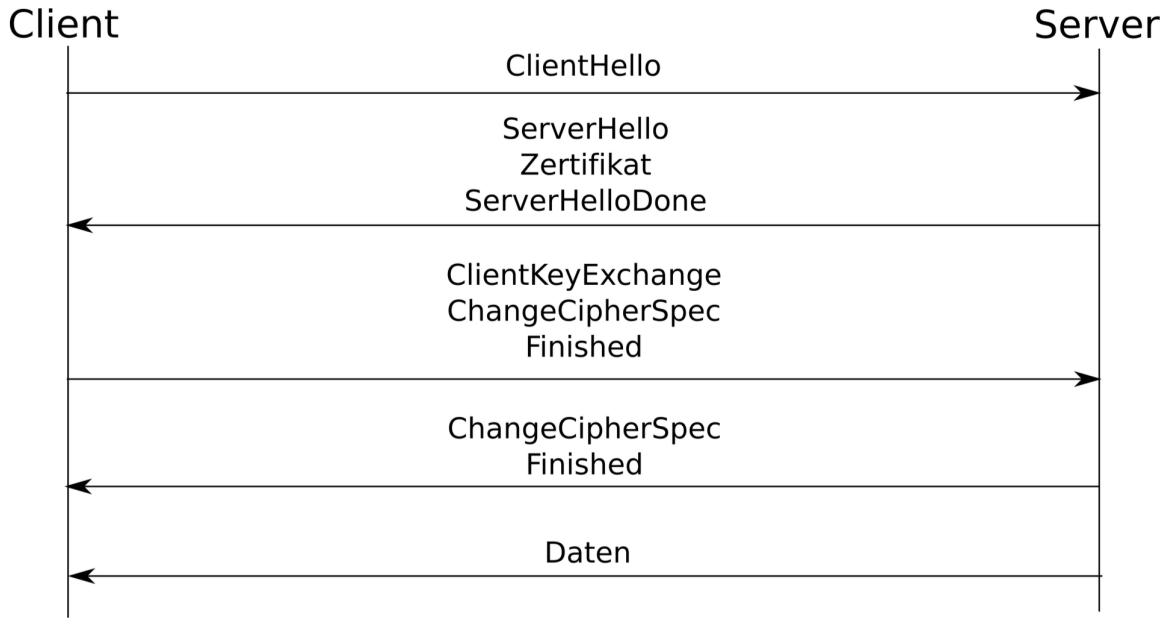
Transport Layer Security (TLS) ist ein kryptografisches Protokoll, das sichere Kommunikation über Netzwerke durch Verschlüsselung, Authentifizierung und Datenintegrität gewährleistet.

Ziele

1. **Vertraulichkeit:** Verschlüsselt die Datenübertragung, sodass Dritte sie nicht lesen können.
2. **Integrität:** Stellt sicher, dass die Daten während der Übertragung nicht manipuliert wurden.
3. **Authentifizierung:** Verifiziert die Identität der Kommunikationspartner (oft durch digitale Zertifikate).

Bestandteile

1. **Handshake:** Aushandlung von Algorithmen, Schlüsseln und Zertifikaten (Cypher-Suite) vor Beginn der Datenübertragung.
2. **Verschlüsselung:** Erster Schlüsseltausch mit **asymmetrischer Verschlüsselung**. Nach Handshake, **symmetrischer Verschlüsselung** für die Datenübertragung.
3. **Zertifikate:** Digitale Zertifikate (PKI) gewährleisten die Authentizität des Servers (und ggf. des Clients).



Schlüsselaspekte

- **Protokolle:** TLS ersetzt das unsichere SSL (Secure Sockets Layer).
- **Port:** Standardmäßig auf Port 443 für HTTPS (HTTP über TLS).
- **Versionen:** Neueste Version ist **TLS 1.3**, seit 2018.

Anwendung

TLS wird in Web-Browsern, E-Mail, VoIP, VPNs und anderen Kommunikationsprotokollen verwendet.

Vorteile

- schützt vor MiM Angriffen
- ermöglicht sichere Online-Transaktionen und Datenübertragung

Probleme

- Zertifikat verwendet veralteten Hashing-Algorithmus
- Zertifikat stimmt nicht mit DNS-Namen des Servers überein
- Abgelaufenes TLS-Zertifikat

Netzwerksicherheit

Beispiele Technische/nicht technische Angriffe

Vertraulichkeit

- Google Hacking
- Covert Channels
- Sniffing

Integrität, Authentizität, Nichtabstreitbarkeit

- Spoofing

Verfügbarkeit

- Session Hijacking
- DOS (Denial of Service)
- DDOS (Distributed Denial of Service)

Google Hacking

Es werden verschiedene Suchbegriffe bei Google verwendet um z.B. Websites mit Passwörtern im Source Code zu finden

Covert Channels

Covert Channel sind versteckte Kanäle, die unauffällig für die Übertragung von Informationen verwendet werden. Beispielsweise kann man Informationen verstecken, indem man TCP SYN-Pakete in unterschiedlichen Intervallen an ein Ziel schickt und das Ziel diese Informationen dann als binäres Signal 0/1 interpretiert oder dass Informationen in einem HTTP GET-Request versteckt werden.

Sniffing

Sniffing heißt, dass man Netzwerk Traffic belauscht und aufzeichnet. Diesen kann man dann analysieren und vertrauliche Daten lesen.

Spoofing

Angriff auf Integrität, Authentizität und Nichtabstreitbarkeit. Spoofing bedeutet das Vortäuschen von Informationen. Im Netzwerk-Bereich kann man beispielsweise eine fremde MAC-Adresse oder die IP-Adresse vortäuschen. Wir werden später noch detaillierter darauf eingehen.

Session Hijacking

Session Hijacking bedeutet, dass eine fremde Session übernommen wird.
Auf Netzwerk-Ebene könnte beispielsweise eine TCP-Session übernommen werden.
Auf Applikationsebene eine Login-Session bei einer Webapplikation

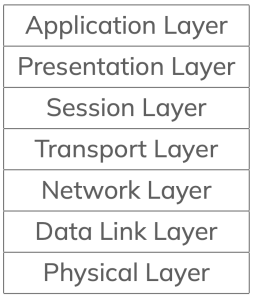
DOS/DDOS

DoS bedeutet, dass Angreifer:innen versuchen ein Service für Berechtigte so zu stören, dass diese es nicht mehr nutzen/nicht mehr darauf zugreifen können. DoS-Angriffe können durch viele parallele bösartige Zugriffe stattfinden, sodass das Service überlastet wird, oder durch Fehler in der Implementierung. Bei letzterer Variante kann schon ein einzelner Zugriff ausreichen, z.B. wenn ein zu langer Eingabe String ein Service zum Absturz bringt.
Wenn DoS-Angriffe gleichzeitig von vielen unterschiedlichen Quellen auf ein Ziel durchgeführt werden, spricht man von Distributed Denial of Service-Angriffen.

ISO/OSI Modell

Das ISO/OSI-Modell beschreibt ein standardisiertes Schichtenmodell für die Kommunikation in Netzwerken.

- **ISO** – International Organization for Standardization
- **OSI** – Open Systems Interconnection



Aspekte

- Reduzierung der Komplexität der Abhängigkeiten
- Trennung der Aufgaben in einzelnen Schichten
- Schichten weitgehend unabhängig voneinander
- Genau definierte Schnittstellen zwischen den Schichten
- Höhere Schichten greifen auf Funktionen niedrigerer Schichten zu

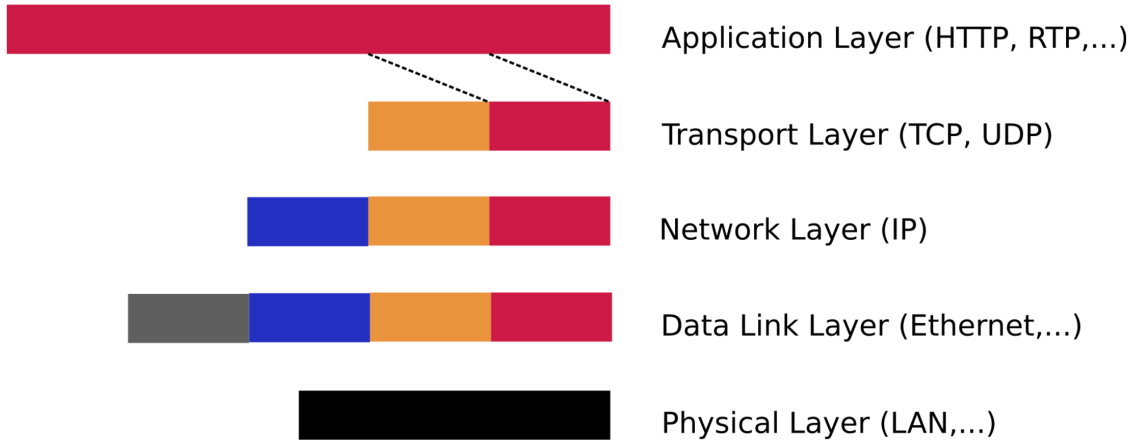
Auswirkungen auf die IT-Sicherheit

1. **Klare Schichten:** Angriffe können gezielt einer Schicht zugeordnet werden (z. B. ARP-Spoofing auf Layer 2, SQL-Injection auf Layer 7).

2. **Schichtenübergreifende Sicherheit:** Schutzmaßnahmen auf einer Schicht können Schwächen auf anderen Schichten nicht automatisch ausgleichen.
3. **Definierte Schnittstellen:** Angriffe nutzen oft Schwachstellen in den Schnittstellen zwischen Schichten aus (z. B. Datenintegrität zwischen Layer 4 und Layer 7).
4. **Unabhängigkeit der Schichten:** Sicherheitsmechanismen sollten auf mehreren Schichten implementiert werden, um Redundanz und Defense-in-Depth zu gewährleisten.

TCP/IP Schichtenmodell

Hat im Gegensatz zu ISO/OSI nur vier Schichten. Dabei werden die ISO/OSI-Schichten 5-7 zur Applikationsschicht und die Ebenen 1-2 zur Netzzugangsschicht zusammengefasst. Zusätzlich gibt es noch die die Transport- und Internetschicht.



ARP (Address Resolution Protocol)

Das **Address Resoultion Protocol (ARP)** übersetzt logische IP-Adresse (ISO/OSI Layer 3) in physische MAC-Adressen (ISO/OSI Layer 2).

Ablauf und ARP-Cache

1. Ein Gerät, das die MAC-Adresse zu einer IP-Adresse rausfinden will, sendet eine Anfrage per Broadcast an alle Geräte im lokalen Netzwerk.
2. Die Geräte, welche die Adressenzuordnung zur IP-Adresse kennen Antworten mit dessen MAC-Adresse.
3. Die zuletzt erhaltene IP/MAC-Adressenzuordnung wird im **ARP-Cache** gespeichert, um erneute ARP-Requests für diese IP-Adresse zu vermeiden.

Schwachstelle und Angriffe

ARP hat **keinen Authentitäts- oder Integritätsschutz.** Angreifer:innen können falsche ARP-Replies an ihr Opfer senden, selbst wenn es keine Anfrage für die IP-Adresse gestellt hat.

ARP Poisoning

Angreifer:innen können den ARP-Cache "vergiften", da er einfach die zuletzt erhaltene Adressenzuordnung übernimmt. Es gibt zwei Arten von ARP Poisoning:

- **Black Hole:** Angreifer:innen verknüpfen ihre eigene MAC-Adresse mit einer Ziel-IP-Adresse. Der Quell-Host sendet Pakete an die Angreifer:innen, die diese verwerfen, wodurch die Pakete "verschwinden".
- **Man in the Middle (MitM):** Angreifer:innen leiten Pakete an sich selbst um, lesen sie mit, verändern sie ggf. und senden sie weiter. Der Empfänger kann diesen Angriff ohne zusätzliche Sicherheitsmaßnahmen nicht erkennen.

Spoofing von MAC-Adressen

Angreifer:innen können leicht beliebige MAC-Adressen imitieren, indem sie sich durch ARP-Poisoning in die Kommunikation zwischen zwei Geräten schalten (MitM). Dies ist z.B. mit Linux-Systemen oder Home-Routern möglich.

ARP-Spoofing

Der Angreifer nutzt eine Schwachstelle des ARP-Protokolls: **ARP-Replies sind nicht authentifiziert** und können ohne vorherigen ARP-Request gesendet werden:

1. **Gefälschte ARP-Replies senden:**Angreifer sendet ein manipuliertes ARP-Paket, das die IP-Adresse von Rechner 2 mit seiner eigenen MAC-Adresse verknüpft.
2. **ARP-Cache manipulieren:**Rechner 1 speichert die falsche Zuordnung im ARP-Cache (IP von Rechner 2 → MAC des Angreifers).
3. **Angriffs-Tools nutzen:**Tools wie **arpspoof**, **ettercap** oder **Scapy** erzeugen und senden gezielt gefälschte ARP-Pakete.

Angriffe auf höhere Protokollschichten

Ähnliche Man-in-the-Middle-Angriffe sind auf Layer 3 (IP) oder Layer 4 (TCP/UDP) möglich, aber sie nutzen andere Techniken wie **DNS-Spoofing** oder **BGP Hijacking**.

IP (Internet Protocol)

Das **Internet Protocol (IP)** ist das grundlegende **Routing-Protokoll des Internets**, das auf ISO/OSI-Layer 3 (Netzwerkschicht) arbeitet. Es ermöglicht die Adressierung und das Routing von Datenpaketen zwischen Netzwerken und wurde erstmals in RFC 791 spezifiziert.

Implementierung:

- **Logische Netzwerkadressen:** IP verwendet Adressen, die Hosts eindeutig identifizieren und konfigurierbar sind.
- **Routing:** IP entscheidet mithilfe von Routing-Tabellen, welche Route ein Paket nimmt, um das Zielnetzwerk zu erreichen. Dabei wird das Paket von Router zu Router weitergeleitet, bis es sein Ziel erreicht. Jeder Router trifft lokal eine Entscheidung basierend auf dem Ziel der IP-Adresse.

Merkmale:

- **Unzuverlässig:** Es wird nicht überprüft, ob Pakete auf dem Weg verloren gehen oder beschädigt werden.
- **Verbindungslos:** Vor dem Versand wird nicht geprüft, ob der Empfänger erreichbar ist oder Pakete annehmen kann.
- Pakete können in **unterschiedlicher Reihenfolge** beim Empfänger ankommen.

Wenn solche Kontrollen/Eigenschaften erforderlich sind, müssen sich Protokolle in anderen Schichten darum kümmern (z.B. TCP).

Adressenformate

- **IPv4:** 4er-Tupel, z.B. 192.168.1.1, weit verbreitet.
- **IPv6:** Nachfolger von IPv4 mit einem größeren Adressraum und zusätzlichen Sicherheitsfunktionen (bereits verwendet, aber noch nicht so weit verbreitet).

Adress-Typen

- **Host-Adressen:** Identifizieren einzelne Hosts.
- **Broadcast-Adressen:** Senden Pakete an alle Hosts eines Netzwerks.
- **Besondere Adressen:**
  - **127.0.0.1:** Adresse für localhost. Diese Adresse ist nur lokal vom Host aus erreichbar und dient zur Selbstadressierung eines Rechners.

Angriffe

Der **IP-Header** enthält wichtige Informationen, die für die Übertragung von Paketen über Netzwerke erforderlich sind. Prinzipiell kann ein Host *alle Felder* manipulieren. Wichtige Felder für Angriffe sind:

- **Source Address:** Gibt die Quell-IP-Adresse an.
- **Destination Address:** Gibt die Ziel-IP-Adresse an.
- **Time to Live (TTL):** Gibt die maximale Anzahl von Hops (Zwischenstationen) an, bevor ein Paket verworfen wird.

Address Spoofing:

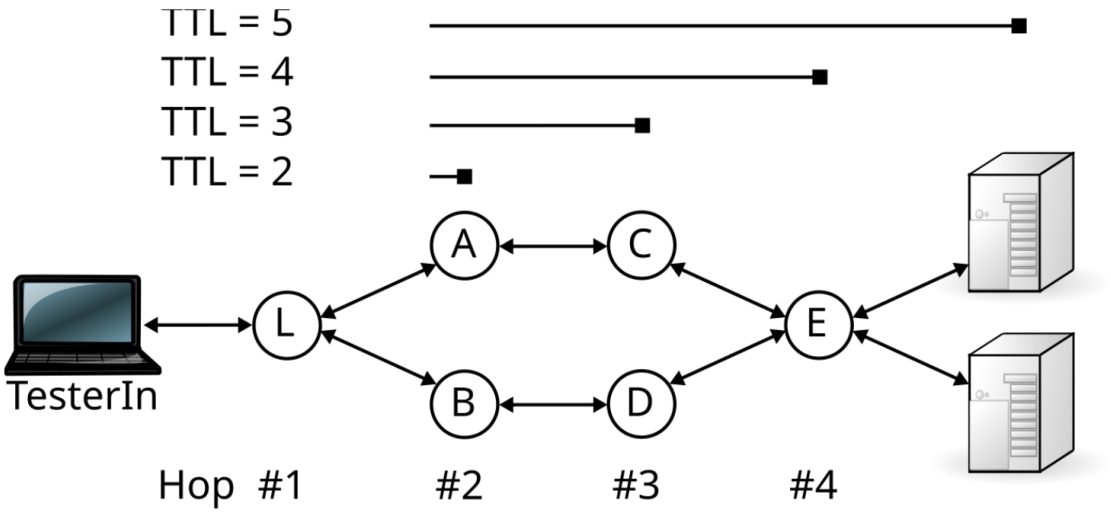
Der Angreifer fälscht die im IP-Header eines Pakets, sodass es aussieht, als käme das Paket von einer anderen IP-Adresse. Das ermöglicht **Denial-of-Service (DoS)** Attacken, bei denen der Angreifer den Server mit einer Flut an Anfragen von gefälschten Adressen überlastet.

Schutz vor Address Spoofing

- Ingress filtering
- Firewalls

ICMP (Internet Control Message Protocol)

Das **Internet Control Message Protocol (ICMP)** ist sowohl nützlich als auch anfällig. Mit dem Befehl `traceroute` (ICMP-basiert) ist es möglich Netzwerkpfade nachzuvollziehen, was die Netzwerkdiagnose unterstützt. Allerdings kann ICMP auch für Angriffe, wie **Smurf-Attacken** missbraucht werden.



Nützlichkeit: Traceroute

`traceroute` ist ein ICMP-basiertes Diagnosewerkzeug, mit dem Netzwerkpfade von IP-Paketen nachvollzogen werden können:

1. Traceroute sendet Pakete mit einem **TTL-Wert** von 1.
2. Der erste Router dekrementiert den TTL-Wert auf 0 und schickt eine **ICMP Time Exceeded** Meldung zurück. Dadurch wird die IP-Adresse des ersten Hops sichtbar.
3. Der TTL-Wert wird schrittweise erhöht (+1), und das Paket wird erneut gesendet, bis das Ziel erreicht wird.

So wird der Weg über alle Hops (Router) hinweg bis zum Zielsystem ermittelt. Die Routen können sich aber zu unterschiedlichen Zeitpunkten unterscheiden, da Routing dynamisch ist.

Missbrauch: ICMP Smurf Attacke

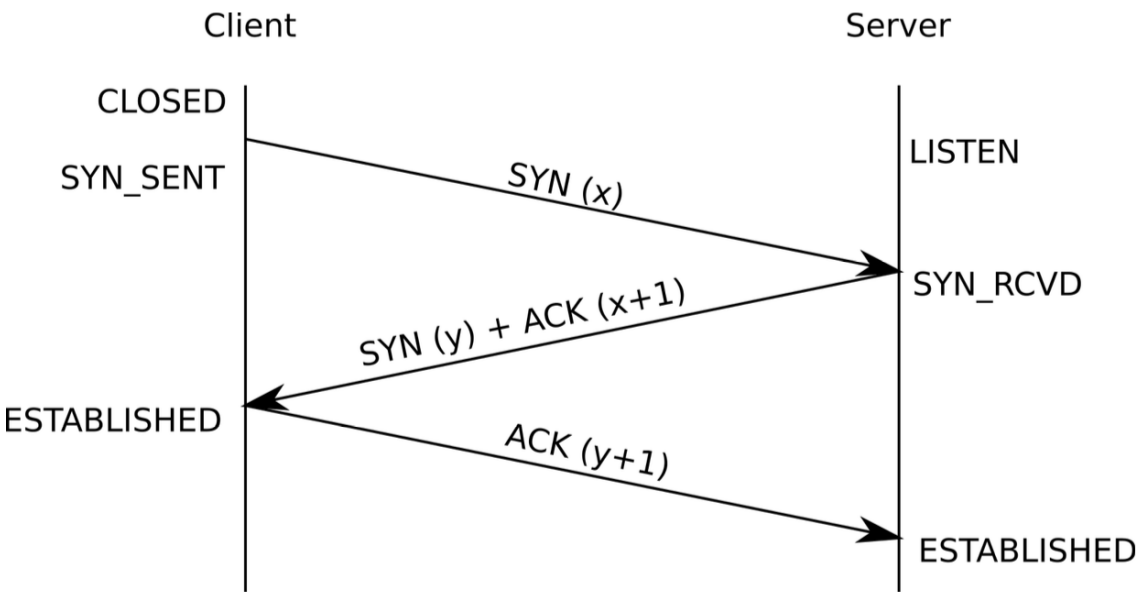
Ein Angreifer nutzt **ICMP Echo Requests** und **IP Address Spoofing**, um ein Opfer mit Paketen zu überschwemmen:

1. **Address-Spoofing**: Die Angreiferin erstellt einen `ICMP Echo Requests` und ändert die **Source Address** des Requests auf die IP-Adresse des Opfers.
2. **Amplification:** Die Angreiferin sendet den verfälschten `ICMP Echo Requests` (Ping) an die **Broadcast-Adresse** eines Netzwerks, die die Anfrage an alle Host-Adressen im Netzwek weiterleitet.
3. **Denial-of-Service**: Alle Hosts im angesprochenen Netzwerk beantworten das Paket mit einem `ICMP Echo Reply` (Pong) und überschwemmen das Opfer mit der Source Adresse.

Heutzutage sind viele Router- und System-Firewalls so konfiguriert, dass sie keine **ICMP Echo Requests** an Broadcast-Adressen beantworten.

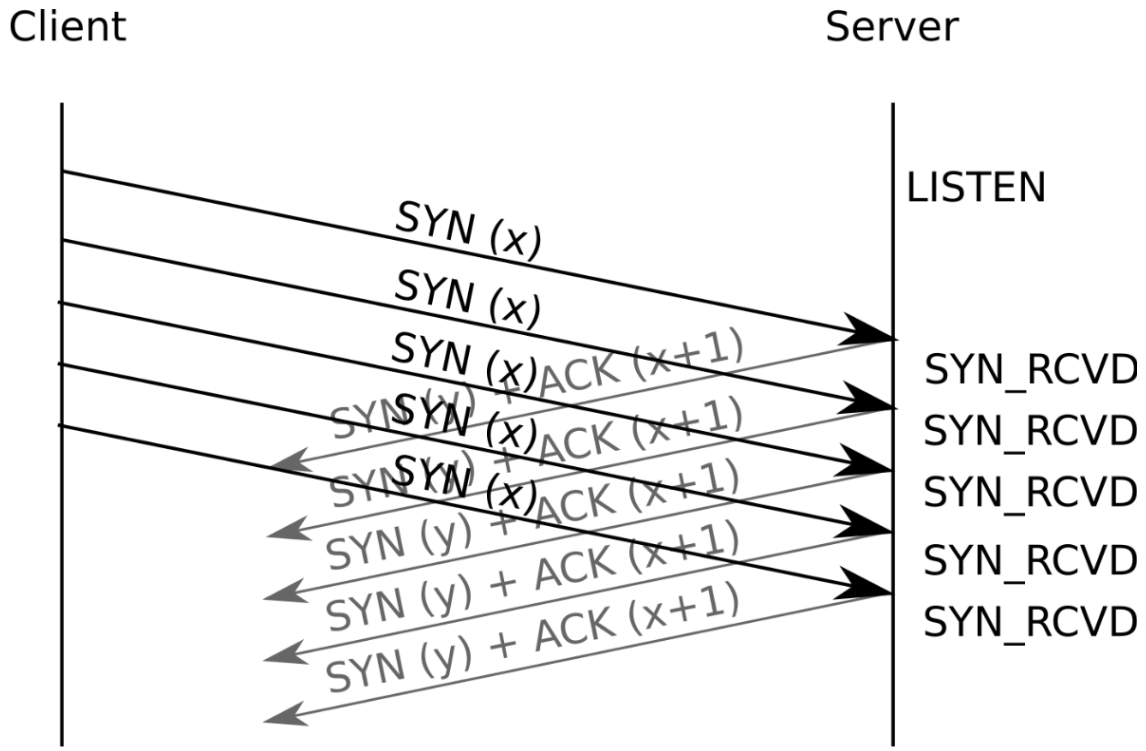
TCP (Transmission Control Protocol)

Das Transmission Control Protocol (TCP) ist ein verbindungsorientiertes und zuverlässiges Protokoll, das mithilfe eines "Three-Way-Handshake" eine stabile Verbindung zwischen zwei Hosts aufbaut und sicherstellt, dass alle Datenpakete in der richtigen Reihenfolge beim Empfänger ankommen.



SYN-Flooding

Hierbei sendet ein Angreifer viele SYN-Anfragen ohne die Absicht, die Verbindung abzuschließen, was den Speicher des Servers überlasten kann. Zum Schutz können SYN-Cookies verwendet werden, die keine zusätzlichen Speicherressourcen für unvollständige Verbindungen benötigen



QUIC

- baut auf User Datagram Protocol (UDP) auf
- zuverlässig, verbindungsorientiert, verschlüsselt; Connection ID zur Identifizierung von Verbindungen
- verwendet TLS zur Absicherung der Kommunikation
- Optimierung auf Performance
- Multiplexing, Retransmission/Flow Control je Stream\
- Verschlüsselung von Nutzdaten und auch einiger Teile des Header

Vorbereitung Angriff

- **Host Scan:** alle verfügbaren Hosts erkennen
- **Port Scan:** kann zeigen, welche Services Aktiv sind
- **OS Detection:** versucht beispielsweise durch Unterschiede in der Implementierung des TCP/IP Netzwerk Stacks herauszufinden, welches Betriebssystem verwendet wird
- **Vulnerability Scan:** Versucht die Version eines Services herauszufinden & bekannte Vulnerabilities auszunutzen

WLAN

Wireless Netzwerke sind potenziell unsicher, da Angreifer:innen auch von einer gewissen Entfernung Zugriff auf das Übertragungsmedium haben. Früher war das sogenannte "War Driving" bekannt, bei dem man nach unverschlüsselten oder schlecht verschlüsselten (WEP) WLAN-Netzwerken gesucht und diese teilweise auch in (Stadt-)Plänen veröffentlicht hat.

Angriffe

- Sniffing
- Spoofing
- DDOS

Absicherung

- WEP mittlerweile veraltet und unsicher (Verschlüsselung kann innerhalb weniger Minuten geknackt werden)
- WPA & WPA2 gute Kennwörter erforderlich
- WPA3
- VPNs
- Standard-Passwörter ändern

Firewalls

- Ziel: Unterbindung von unerlaubten Zugriffen
- Positionierung von Firewalls i.A. an der Grenze zwischen zwei Netzwerkzonen („Zonenmodell“, DMZ)

Arten von Firewalls

- **Paketfilter**: die einfachsten Firewalls. Sie schauen nur auf jedes einzelne Paket und entscheiden basierend auf MAC-Adresse, IP-Adressen und Ports, ob ein Paket von der Firewall geblockt werden soll oder nicht.
- **Stateful Inspection:** kennen den Status einer Verbindung. Beispielsweise weiss eine solche Firewall, ob bereits ein TCP Three-Way-Handshake durchgeführt wurde oder nicht. Hier können daher weitere Entscheidungsgrundlagen für ein Ablehnen eines Pakets verwendet werden.

- **Proxy Firewall:** agiert auf Applikationsebe und leitet Anfragen als Proxy weiter. Sie versteht beispielsweise das HTTP-Protokoll und filtert nicht erlaubte HTTP-Methoden heraus. Beispielsweise könnte man aktivieren, dass nur simple HTTP-GET-Requests erlaubt werden, HTTP-Post-Requests allerdings nicht. Eine Proxy Firewall nimmt Anfragen also entgegen und führt eine neue Anfrage an das Zielsystem durch. Die Antwort wird dann wieder an das ursprüngliche System weitergeleitet.

Zero Trust Architecture (ZTA)

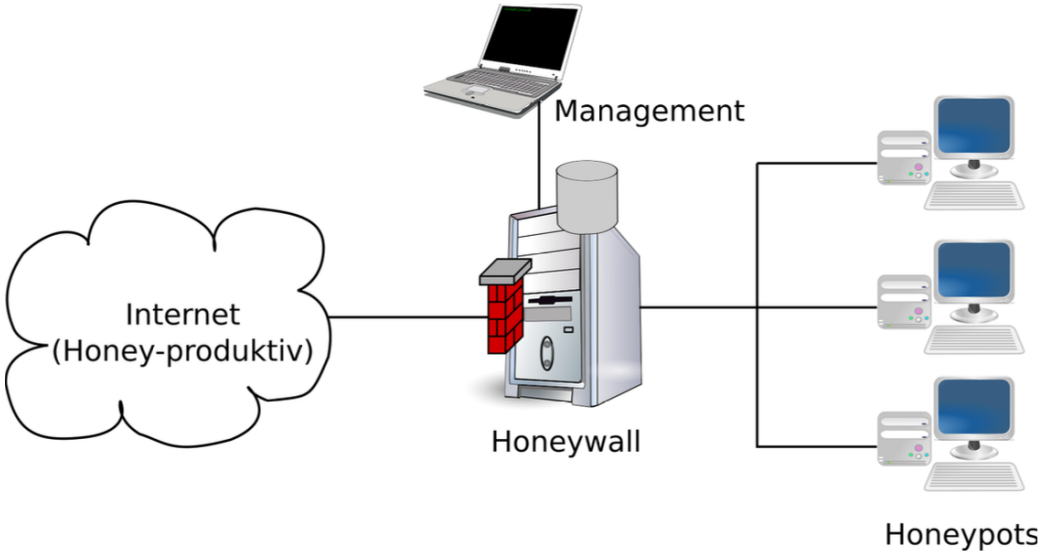
Der Zugriff wird nicht mehr auf Grund einer Netzwerkzugehörigkeit erlaubt, sondern jeder Zugriff wird einzeln bewertet & validiert.

Honeypot und Honeynet

- Kein Produktionssystem
- Kein legitimer Zugriff
- Leichtere Analyse von auftretendem Traffic
- Honeynet ist ein Netzwerk von Honeypots

Definition

- „A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource.“ (Spitzner)
- „A security resource who’s value lies in being probed, attacked or compromised.“ (Spitzner)



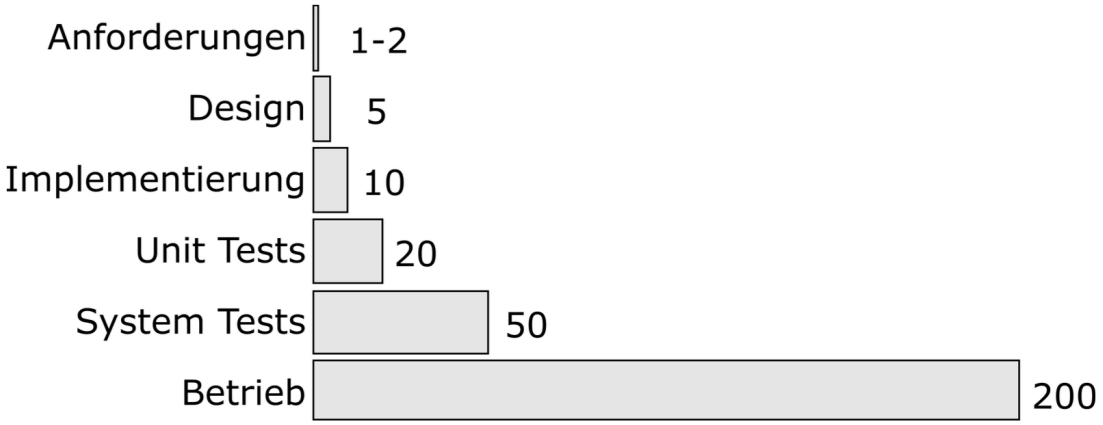
Sicherheit in der Softwareentwicklung

Entwicklungsphasen

- Analyse/Anforderungen
- Design
- Implementierung
- Test
- Betrieb

"Sicherheit ist ein Prozess" *(Bruce Schneier)* und kein Feature, das einmalig Anwendung in einer der SDLC Phasen findet.

Relative Kosten von Softwarefehlern



Sicherheit in der Analysephase

Als Analyse wird die **Erhebung und Aufbereitung der Sicherheitsanforderungen** für das **zu entwickelnde System** bezeichnet.

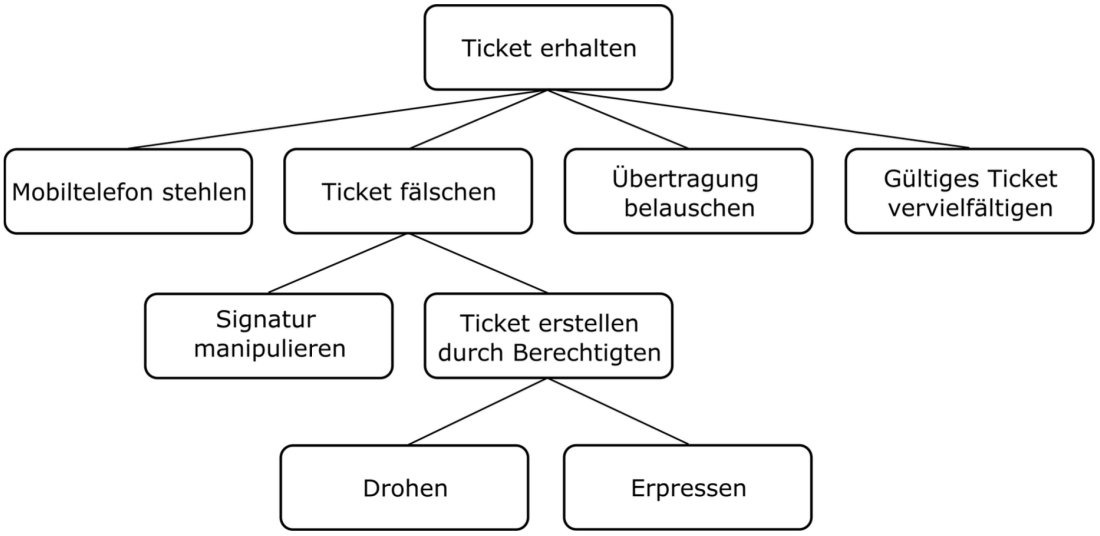
Durchführung der Analyse

- Ermittlung von Bedrohungen & Risiken
- Vollständige Erfassung der Angriffsoberfläche
- Festlegung der Schutzziele für die verarbeiteten Daten
- Systematisches Vorgehen zur vollständigen Erfassung erforderlich
- Werkzeuge zur Durchführung vorhanden, z.B. Angriffsbäume oder Threat Modelling

Angriffsbäume/Attack Trees

Angriffsbäume dokumentieren Bedrohungen in einer Baumstruktur.

- Jeder Knoten stellt eine Bedingung dar, die erfüllt werden muss, um das Ziel (Wurzel) zu erreichen.
- Sie können iterativ detailliert werden und Gewichtungen (z. B. Eintrittswahrscheinlichkeiten) enthalten.



Threat Modeling

Methode von Microsoft zur Modellierung von Bedrohungen in der Software:

- frühzeitige Beschäftigung mit möglichen Bedrohungen
- schrittweise Modellierung
- Bedrohungen identifizieren - Hineinversetzen in Angreifer

Sicherheit in der Entwurfsphase

Entwurf soll Sicherheitsanforderungen abbilden, erstellen einer robusten Architektur, inkludieren von Best Practices:

- **Sicherheitsmuster (Security Patterns)** stellen für immer wieder auftretende Herausforderungen im Sicherheitsbereich Entwurfslösungen zur Verfügung
- **Angriffsmuster (Attack Patterns)** Beschreibung von Angriffen inklusive der Vorbedingungen welche erfüllt werden müssen. Ableitung von Sicherheitsaspekten für das eigene System möglich
- **UML** Spezielle Erweiterungen zur Modellierung von Sicherheitsanforderungen vorhanden z.B. UMLsec

Design Principles

1. Einfachheit der Schutzmechanismen

    - Keep it simple
    - Design komplex → Fehleranfällig
    - Kleine, einfache Codeteile können besser getestet werden
2. Fail-Safe Defaults
    - In der Standard-Einstellung in einem sicheren Zustand sein und im Fehlerfall auf einen sicheren Zustand zurück gehen
3. Vollständige Berechtigungsprüfung (Complete Mediation)
4. Offenes Design (Open Design)
5. 4-Augen-Prinzip (Separation of Privilege)
6. Minimum an Rechten (Least Privilege)
7. Minimum an gemeinsamen Mechanismen (Least Common Mechanism)
8. Psychologische Akzeptanz (Psychological Acceptability)

Sicherheit in der Implementierung

Eine sichere Softwarearchitektur kann durch eine fehlerhafte Implementierung unsicher werden (z.B. fehlerhafte Zertifikatsüberprüfung).

Eine Software ohne Schwachstellen durch Programmierfehler kann Schwächen durch die Architektur aufweisen.

Sichere Programmierung

- Auflistungen der häufigsten Sicherheitsfehler, z.B.
    - OWASP: Top Ten Vulnerabilities (u.a. Web und Mobile Applikationen)
    - CWE/SANS: TOP 25 Most Dangerous Programming Errors
- Programmierrichtlinien
    - SEI CERT Secure Coding Standards
    - OWASP Developer Guide
- Sicherheitskonzepte von Programmiersprachen
    - Automatische Längenprüfung und Speicherverwaltung, …
    - Kenntnis und korrekte Anwendung über vorhandene Sicherheitskonzepte
    - Verwendung von Sprachen mit Typ-Sicherheit

Sicherheitsprobleme von Artefakten

Durch die Nutzung von Bibliotheken und Frameworks können unbeabsichtigt Sicherheitsprobleme eingebracht werden.
Gegenmaßnahme ist eine
sichere Übernahme der Artefakte, bspw. durch CVE-Tracking

Penetrate and Patch

Sicherheitsfehler, die im Betrieb gefunden und an den Hersteller gemeldet werden, durch Live-Patches beseitigt

Für sichere Software nicht ausreichend:

- Ungemeldete Fehler können nicht ausgebessert werden
- Patch bessert nur Symptom nicht Ursache aus
- Zeitfenster für Angreifer bis überall Patch eingespielt
- Ableiten von Schwachstellen durch Patches

SQL Injections

Eine Manipulation des SQL Statements durch eine/n Angreifer:in ist möglich:

- Einfaches Hochkomma: te'st
- Fehlermeldung der Datenbank wird angezeigt
  - Schwachstelle erkennbar.
  - Zusätzlich werden u.U. Informationen zur Datenbank und zum Aufbau des SQL Statements ausgegeben.
- Verwendung von Tautologien wie ' or '1'='1 was zu SELECT * FROM users WHERE password='' or '1'='1' führt
- Auslesen von Daten aus weiteren Tabellen mittels UNION oder Sub Queries
  - ' UNION SELECT * FROM system

Gegenmaßnahmen:

- Eingabevalidierung (Allowlist bevorzugt)
- Verwendung von Prepared Statements
- Minimierung der Datenbankrechte (Principle of Least Privilege)

Command Injections

Wie bei SQL Injection wird eine Eingabe ohne ausreichender Validierung für einen Systemaufruf verwendet. Dabei besteht die Möglichkeit den Aufruf zu manipulieren und beliebige Programme auszuführen.

Gegenmaßnahmen:

- Validierung der Eingaben!
- Verwendung der Funktionen von Programmiersprachen zum Lesen von Dateien
- Beschränkung der Zugriffsrechte auf Dateien durch Rechtevergabe oder Verwendung von chroot Umgebungen
- Minimierung der möglichen ausführbaren Dateien
- Berücksichtigung der System-Konfigurationen (z.B. sichere PHP Konfiguration)

Cross-Site-Scripting (XSS)

Bei XSS wird unvalidierte Benutzereingaben durch die Software ausgegeben, wodurch Schadcode auf einer vertrauenswürdigen Webseite ausgeführt wird.

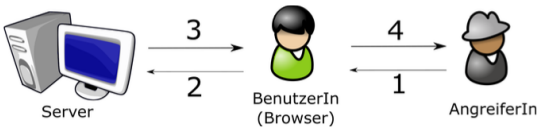
- Reflected XSS: Schadcode wird über URL eingeschleust.
- Stored XSS: Schadcode wird persistent gespeichert.

Gefahr von Session-Hijacking, Überschreiben der Website durch Angreifer, Weiterleitung von Nutzern

Gegenmaßnahmen:

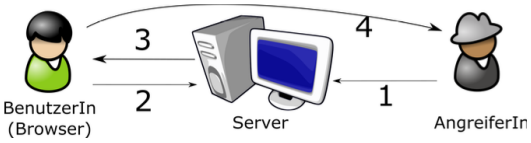
- Validieren *aller* Eingaben
- HTML-Encoding von Benutzereingaben
- Client-seitige Verhinderung durch Browser oder Client-Firewalls

Cross-Site-Scripting (XSS) – „Reflected“



1. Angreifer:in sendet präparierte URL an Benutzer:in über weiteren Kanal (z.B. E-Mail)
2. Benutzer:in ruft Seite im Browser auf
3. Server liefert reguläre Seite mit zusätzlichen Inhalten aus präparierter URL zurück
4. Browser wertet Rückgabe von Server aus. Je nach Angriff sofort Übermittlung von Daten an Angreifer oder umleiten nach weiteren Eingaben des Benutzers/der Benutzerin zum Angreifer/zur Angreiferin

Cross-Site-Scripting (XSS) – „Stored“



1. Angreifer:in fügt präparierten Inhalt ein. Persistente Speicherung des Inhaltes durch Server
2. Benutzer:in ruft Seite im Browser auf
3. Server liefert reguläre Seite mit zusätzlichen präparierten Inhalten des Angreifers/der Angreiferin zurück
4. Browser wertet Rückgabe von Server aus. Je nach Angriff sofort Übermittlung von Daten an Angreifer:in oder umleiten nach weiteren Eingaben des Benutzers/der Benutzerin zum Angreifer/zur Angreiferin

Cross-Site-Request-Forgery (CSRF)

CSRF ist eine Design-Schwachstelle einer Webanwendung. Durch untergeschobene URL kann Angreifer Benutzer für Aktion am Server missbrauchen

Gegenmaßnahmen:

- Verwendung von Shared Secrets (z.B. geheimes CSRF-Token)

- Benutzeraktionen verifizieren (z.B. „Wollen Sie das wirklich?“)

Buffer Overflows

Ein Buffer Overflow entsteht durch Programmierfehler, wenn Eingaben ohne korrekte Längenüberprüfung in den Speicher geschrieben werden.
Auswirkungen: Absturz, ungewöhnliches Verhalten, Ausführung von Schadcode

Gegenmaßnahmen:

- Korrekte Input Validierung und Längenüberprüfung
- Verwendung von Programmiersprachen mit automatischer Längenüberprüfung
- Vermeidung von gefährdeten Funktionen (strcpy(), gets(), strcat(),…)
- Verwendung von Testmethoden zum Finden von Buffer Overflows im Quellcode (Statische Codeanalyse, Fuzzing,…)

Sicherheit in der Betriebsphase

- Es zeigt sich, ob Sicherheitsanforderungen tatsächlich erfüllt werden
- Kenntnis von Sicherheitsanforderungen für den Betrieb
- Definition und Einhaltung des Service Levels (Service Level Agreement (SLA))
- Bemerken von Sicherheitsvorfällen
- Mögliche Reaktionen auf Sicherheitsvorfälle
- ITIL (Information Technology Infrastructure Library)
- Organisatorische Sicherheitsanforderungen

Identität, Authentifizierung, Zugriffskontrolle

Begriffe

- **Identität:** Wer jemand ist
- **Authentisierung („authentication“)**: Vorlage eines Nachweises zur Identifikation (z.B. Username/Passwort)
- **Authentifizierung („authentication“):** Überprüfung eines Nachweises zur Identifikation

- A**utorisierung („Authorization“):** Überprüfung, ob eine Person, IT-Komponente oder Anwendung zur Durchführung einer bestimmten Aktion berechtigt ist („Rechtetabelle“)
- **Zutritt:** Betreten von abgegrenzten Bereichen wie z.B. Räumen oder geschützten Arealen in einem Gelände
- **Zugang:** Nutzung von IT-Systemen, System-Komponenten und Netzen
- **Zugriff:** Nutzung von Informationen bzw. Daten

Gründe für Authentisierung/Authentifizierung

- Nachweis der Identität von Personen, Diensten oder Komponenten.
- Ermöglicht Kontrollen von Zutritt, Zugang und Zugriff.
- Dient der Zurechenbarkeit und Nachvollziehbarkeit von Aktionen.
- Stellt Vertrauen zwischen zwei Kommunikationspartnern her

Methoden der Authentisierung

1. Wissen (z. B. Passwort, PIN)
2. Besitz (z. B. Token, Chipkarte)
3. Biometrische Merkmale (z. B. Fingerabdruck, Iris-Scan)

Authentisierung durch Wissen (Passwörter)

Vorteile:

- Kostengünstig und etabliert
- Passwörter können individuell geändert werden
- Benutzer sind mit Passwörtern vertraut

Nachteile:

- Verlust ist schwer zu erkennen.
- Niedrige Passwort-Komplexität erleichtert Angriffe.
- Sicherheitsfragen sind oft leicht zu erraten (z. B. Name des Haustiers).
- Wiederholtes verwenden von Passwörtern

Regeln beim Verwenden von Passwörtern

- Keine Default-Passwörter verwenden
- Passwörter sollen nicht aufgeschrieben werden (v.a. nicht am Monitor, Keyboard etc.)
- Passwörter sollen von Benutzer:innen änderbar sein
- Anzahl der Versuche zur Passworteingabe beschränken (Achtung: Denial of Service)

Angriffe auf Passwörter

1. Raten von Passwörtern
2. Brute-Force-Angriffe (Wörterbuch, Kombinationen)
3. Social Engineering
4. Rainbow-Tables (ohne Salt)
    - Listen mit Passwörtern und vorberechneten Hashes.
    - Diese können genutzt werden, wenn Passwörter als Hash gespeichert werden, aber über keinen Salt verfügen.
    - Ein Salt ist ein pro Passwort zufällig generierter String, mit dem das Passwort vor dem Hashen verknüpft wird.
    - Dieser Salt muss nicht unbedingt vertraulich sein.
    - Er dient nur dazu, dass die selben Passwörter bei unterschiedlichen User:innen bei gleichem Hash-Verfahren einen anderen Hashwert erhalten.
5. Sniffing (z. B. über Netzwerke)
6. "Über die Schulter Schauen"
7. Preisgabe von Information: „Username/Passwort falsch“

Abwehrmethoden Angriffe auf Passwörter

- Passwörter müssen am System sicher gespeichert werden, sie sollen nicht im Klartext gespeichert werden (Hash, Salt), Zugriffsschutz
- Eigene Hash-Algorithmen mit bestimmten Eigenschaften zur Verarbeitung von Passwörtern (siehe Provos und Mazières bzw. Password Hashing Challenge), z.B. Argon2, bcrypt
- Proaktive Passwort-Checker

- User-Training
- Lockout Mechanismen
- Passwort Wechsel bei Verdacht auf Security-Incidents
- Fehlanmeldungen/Letzte erfolgreiche Anmeldung anzeigen
- Verwendung von Passwort-Safes mit automatisch erstellten Passwörtern

Authentisierung durch Besitz

Besitz bedeutet, dass für die Authentisierung ein physischer Gegenstand (z. B. Token, Chipkarte) vorgelegt werden muss.

Beispiele:

- Speicherung kryptographischer Schlüssel
- Chipkarten mit kryptographischen Schlüsseln, die durch PINs geschützt sind.
- Erzeugung von Einmal-Passwörtern basierend auf Zeit oder einem Counter

Vorteile:

- Verlust leichter bemerkbar
- Schwer zu kopieren (?)

Nachteile:

- Teurer als Passwörter weil physischer Token erstellt werden muss

Angriffe auf Chipkarten

Eine Chipkarte speichert kryptographische Schlüssel sicher und führt kryptographische Operationen direkt auf der Karte aus. Der Zugriff ist meistens durch einen PIN-Code geschützt.

- Social Engineering
- Man-in-the-Middle-Angriffe
- Stromverbrauchs- oder Rechenzeitanalyse
- Physikalische Manipulation (z. B. Temperatur, Spannung)

Authentisierung durch biometrische Merkmale

Verwendung von Fingerabdrücken, Gesichtserkennung, Stimme, Iris-Scan.

Vorteile:

- Schwer zu ersetzen oder weiterzugeben.

Nachteile:

- Fehlerquoten
  - FAR (False Acceptance Rate): Wie oft Angreifer akzeptiert werden.
  - FRR (False Rejection Rate): Wie oft berechtigte Benutzer abgelehnt werden.
- höhere Kosten
- Datenschutzbedenken

Angriffe auf biometrische Merkmale

- Authentifizierung nicht zu 100% sicher
- Stärken/Schwächen biometrischer Authentisierung werden oft vernachlässigt
- Spoofing (Hochauflösende Fotos (z.B. 31C3), Modellierung von (gefundenden) Fingerabdrücken,…)
- Beispiel: Samsung Galaxy S5: Fingerabdrucksensor auch schon gehackt
- Replay Attacken
- Umgehen der Sensoren
- Brute Force
- Neuartige Angriffe durch KI

2-Faktor/Multifaktor-Authentifizierung

Kombination von zwei oder mehr Authentisierungsmethoden erhöht die Sicherheit.

Beispiele:

- Wissen und Besitz (z.B. Passwort und Token).
- Wissen und Biometrie (z.B. Passwort und Fingerabdruck).

FIDO2 – Fast IDentity Online 2

Technologien wie FIDO2 und Passkeys werden zunehmend eingesetzt, um sichere und benutzerfreundliche Authentifizierungslösungen zu bieten.

Grundidee:

- Pro Site Erzeugung eigenes Private/Public Keypaars
- Public Key kommt bei Registrierung zum Service, Verknüpfung mit Account
- Bei Anmeldung wird Challenge (incl. Anfrage-Domain) zum Authenticator geschickt
- Automatische Auswahl des richtigen Schlüssels
- Signatur nach User-Interaktion
- Nach erfolgreicher Signaturprüfung Anmeldung erfolgreich

Passkeys

Variante von FIDO2

Ziele:

- Ersatz von Passwörtern
- Höhere Sicherheit statt einfacher Passwörter
- Schutz vor Phishing

Varianten:

- Gerätegebunden
- Synchronisierte Passkeys

  ▶️ Übergreifende Verwendung von Passkeys über Devices und Plattformen hinweg
  ▶️ Verschlüsselte Speicherung der Passkeys in Clouds

Zugriffskontrolle

Zugriffskontrolle stellt sicher, dass nur berechtigte Akteur auf Ressourcen zugreifen können. Unberechtigter Zugriff wird verhindert oder erkannt.

- Angreifer:in hat Zugriff auf Binärdaten → Kryptographische Methoden erforderlich
- Es existiert eine Schicht zwischen Angreifer:in und Binärdaten → Zugriffskontrolle

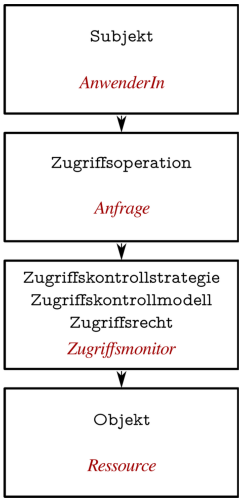
Kernfrage: *Wer* darf auf *was wie* zugreifen?

Zugriffskontrollstrategie

Ein Regelwerk, das besagt was erlaubt/nicht erlaubt ist.

Zugriffskontrollmodelle

- Ein Formalismus, um eine Zugriffskontrollstrategie zu beschreiben.
- Verschiedene Zugriffskontrollmodelle für unterschiedliche Zugriffskontrollstrategien
- Soll einfach, ausdrucksstark, intuitiv, wartbar und umsetzbar sein.



Begriffe

- **Objekt**: Passiver, zu schützender Informationsträger.
- **Subjekt**: Aktive Elemente, die im Auftrag von Anwender:innen Zugriffe auf Informationen ausführen.
- **Zugriffsoperation**: Art, um auf ein Objekt zuzugreifen (z. B. read, write, execute, …).
- **Zugriffsrecht**: Rechte für Zugriffe auf Dateien, Datenträger, Prozesse, etc.
- **Schutzdomäne**: Gruppierung von identischen Zugriffsrechten.
- **Zugriffsmonitor**: Setzt die Zugriffskontrollstrategie durch.

Zugriffsmatrix

Eine Matrix, die Zugriffsrechte eines Subjekts auf ein Objekt definiert. Zeilen repräsentieren Subjekte, Spalten Objekte, und Einträge geben die Berechtigungen an.

		Objekt				
		α	β	γ	δ	ϵ
Subjekt	a	0	0	1	0	1
	b	1	1	1	1	0
	c	0	0	0	0	1
	d	0	1	1	1	0
	e	1	1	0	1	0
	f	0	1	0	0	1

Discretionary Access Control (DAC)

- Besitzer:in eines Objekts legt Zugriffsrechte selbst fest.

- Flexibel, aber schwer zu verwalten bei großen Systemen.
- **Nachteile**:
  - Komplexe Rechteverwaltung.
  - Hohe Fehleranfälligkeit.

Role-Based Access Control (RBAC)

- Zugriffsrechte basieren auf Rollen statt auf individuellen Benutzern.
- Einfachere Verwaltung durch Gruppierung von Rechten.
- Unterstützung von Hierarchien und Separation of Duties.

Mandatory Access Control (MAC)

- Zentrale Festlegung der Zugriffsrechte basierend auf Sicherheitsstufen.
- Strikte Kontrolle des Informationsflusses.
- Hauptsächlich im militärischen Bereich eingesetzt.

Weitere Aspekte der Zugriffskontrolle

- **Security Policies**: Regelwerke, die definieren, was erlaubt ist.
- **Access Control Mechanisms**: Umsetzung der Policies auf verschiedenen Ebenen (z. B. Betriebssystem, Datenbanken, Webserver).
- **Identity and Access Management (IAM)**: Verwaltung von Benutzeridentitäten und Zugriffsrechten.
- **Single Sign-On (SSO)**: Ermöglicht die Nutzung mehrerer Systeme mit einer einzigen Authentifizierung.
- **Attribute-Based Access Control (ABAC)**: Zugriffsrechte basieren auf Attributen der Benutzer und Ressourcen.

Organisatorische Sicherheit & Sicherheitsmanagement

Organisatorische Sicherheitsaspekte

- **Primärziel**: Aufbau des IT-Sicherheits-Managements (Sicherheitsplanung).

- **Geschäftsführung (GF)**: Formulierung einer IT-Sicherheitsleitlinie (strategische Aussagen).
- **Typische Aufgaben**:
  - Eindeutige Definition von Verantwortlichkeiten inklusive Vertretungsregelungen sowie Kommunikationswege.
  - Schulung von IT-Benutzer:innen und Sensibilisierung für Sicherheit.
  - Dokumentation aller Maßnahmen und Veränderungen.
  - Durchführung regelmäßiger Audits.
- **Grundsatz**:
  - IT-Sicherheit muss von Management, Administrator:innen und IT-Benutzer:innen als fortlaufender Prozess verstanden und gelebt werden.

Beispiele für organisatorische Sicherheitsmaßnahmen

- **Schulung von Mitarbeitern** um Sicherheitsbewusstsein zu schaffen
- **Sicherung von Daten** durch Backups
- **Sicheres** (unwiderrufliches) **Löschen** von Daten
- **Updates** für Sicherheitssoftware
- **Dokumentation** von Sicherheitsprozessen
- Regelmäßige **Überprüfung Wirksamkeit** der Maßnahmen
- **Checklisten**

Security Policies

Security Policies legen Richtlinien, Verantwortlichkeiten und Ziele der IT-Sicherheit fest. Sie müssen klar, prägnant und für die jeweilige Zielgruppe relevant sein. Eine effektive Policy erfordert Unterstützung durch das Management und regelmäßige Überprüfungen auf Umsetzbarkeit und Aktualität.

Anforderungen:

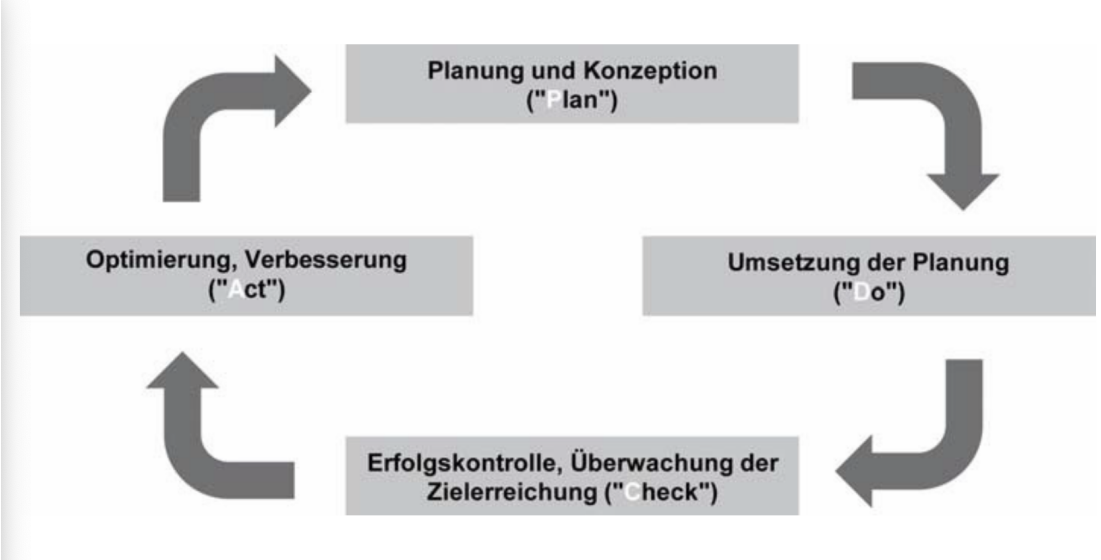
- Vollständigkeit der Policies
- Bekanntheit der Policies
- Verständnis und aktives „Leben“ der Policies

- Laufende Überprüfung der Einhaltung von Policies

Sicherheitsmanagementprozess/PDCA-Zyklus

- **Plan**:
  - Definition der Sicherheitsziele.
  - Festlegung von Verantwortlichkeiten.
  - Erstellung von Sicherheitsrichtlinien.
- **Do**:
  - Implementierung von Sicherheitsmaßnahmen.
  - Durchführung von Schulungen.
  - Monitoring der Systeme.
- **Check**:
  - Regelmäßige Überprüfungen.
  - Durchführung von Audits und Tests zur Bewertung der Wirksamkeit der Maßnahmen.
- **Act**:
  - Anpassung und Verbesserung der Sicherheitsmaßnahmen basierend auf den Ergebnissen der Prüfungen.

Hinweis: Dieser Prozess ist ein kontinuierlicher Kreislauf, der auf neuen Bedrohungen und Erkenntnissen basiert.



Standardisierung und Zertifizierung

Zur Sicherstellung eines hohen Sicherheitsniveaus werden Standards und Normen wie **ISO 27001**, **IT-Grundschutz** und **ITIL** angewandt.

- **ISO 27001**:
  - ISO 27001 regelt die Errichtung, Überwachung und kontinuierliche Verbesserung eines Informationssicherheitsmanagementsystems (ISMS) zur nachhaltigen Sicherstellung von IT-Sicherheit.
- **IT-Grundschutz**:
  - Bietet ein Baukastenprinzip für die Auswahl und Umsetzung von Sicherheitsmaßnahmen.
  - Speziell geeignet für kleine und mittlere Unternehmen.
- **ITIL**:
  - Sammlung von Best Practices zur Optimierung des IT-Betriebs.
  - Beinhaltet auch Sicherheitsmanagement.

Vorteile von Zertifizierungen:

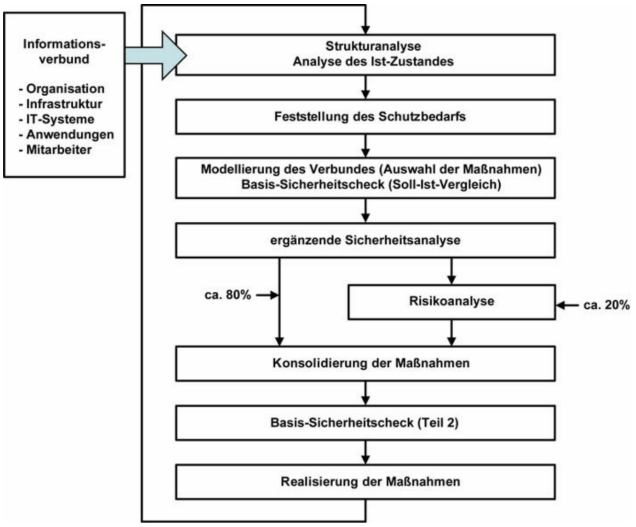
- Stärken das Vertrauen in die IT-Sicherheit einer Organisation.
- Sind oft Voraussetzung für Vertragsbeziehungen.

IT-Sicherheitskonzept

Erforderliche Maßnahmen zur Realisierung und Aufrechterhaltung eines angemessenen und definierten Sicherheitsniveaus

Fragen:

1. Was will ich schützen?
2. Wogegen soll ich mich schützen?
3. Wie kann ich diesen Schutz erzielen?
4. Kann ich mir diesen Schutz leisten?



Business Continuity

Business Continuity beschreibt Verfahren und Konzepte, um die Geschäftstätigkeit bei IT-Ausfällen durch schnelle Wiederherstellung der Systeme sicherzustellen.

Sicherheitsimplikationen von Typisierung in Programmiersprachen

Definition einer sicheren Programmiersprache

1. **Man kann den Abstraktionen vertrauen**, welche die Sprache bietet.
2. **Programme haben immer eine präzise und wohldefinierte Semantik.**
3. **Kein undefiniertes Verhalten**:
    - Sprache verhindert/detektiert unsinnige Ausführung durch Kompilier- oder Laufzeitprüfungen und gibt eine Fehlermeldung aus.

Vorteile und Nachteile eines unsicheren Ansatzes in Programmiersprachen:

- **Vorteile:** Geschwindigkeit und Kompatibilität.
- **Nachteile:** Sicherheitsrisiken und keine Garantie für Programmverhalten.

Safety und (Un)Definedness

- **Safety**: Vermeidung fehlerhafter Operationen wie Speicherzugriffsfehler oder Typkonflikte.

- **(Un)Definedness**: Klarheit über das Verhalten von Statements; undefinierte Semantik führt zu unsicherem Verhalten.
  Beispiel: Das Statement
  `a[i] = (float) x;` ist nur sicher, wenn `a` ein float-Array ist und `i` gültige Indizes besitzt.

Ansätze im Umgang mit undefiniertem Verhalten

1. Akzeptanz undefinierter Semantik mit Verantwortung der Programmierer:innen.
2. Sicherstellung korrekter Ausführung durch die Sprache mittels Prüfungen.
   Sichere Sprachen setzen auf letztere, wodurch fehlerhafte Ausführungen frühzeitig erkannt oder verhindert werden.

Typsicherheit

Typsicherheit bedeutet, dass ein Programm typbedingte Fehler während der Ausführung vermeidet, indem Typregeln strikt eingehalten werden.

Typsichere Sprachen: C#, Go, Haskell, Java, Kotlin, ML, Python, Rust, Swift

Typunsichere Sprachen: C, C++

Typisierungsansätze

- **Statische vs. Dynamische Typisierung**:
  - **Statische Typisierung**: Datentypen werden zur Compile-Zeit geprüft und sind unveränderlich. Fehler werden frühzeitig erkannt (z.B. Java, Kotlin).
  - **Dynamische Typisierung**: Datentypen werden zur Laufzeit geprüft und können sich ändern. Fehler werden erst zur Laufzeit sichtbar (z.B. Python, JavaScript).
- **Starke vs. Schwache Typisierung**:
  - **Starke Typisierung**: Inkompatible Typen müssen explizit konvertiert werden, andernfalls gibt es einen Fehler (z.B. Python, Kotlin).
  - **Schwache Typisierung**: Typkonvertierungen erfolgen implizit, was zu unvorhersehbarem Verhalten führen kann (z.B. JavaScript, PHP).
- **Duck Typing:** Typ wird durch das Verhalten eines Objekts definiert, unabhängig von einer formalen Definition (z.B. Python).
- **Strukturelles Typing**: Typ wird basierend auf der Struktur des Objekts und seinen Mitgliedern überprüft, ohne expliziten Namen (z.B. TypeScript).

- **Nominales Typing**: Typkompatibilität basiert auf dem Namen und der expliziten Definition von Typen (z.B. Java, C#).
- **Typinferenz:** Fähigkeit vom Compiler/Laufzeitumgebung, Typ automatisch aus Kontext zu erschließen, ohne explizite Angabe
- **Generische Typisierung** ermöglicht die Definition von Typen mit Variablen sodass sie mit verschiedenen anderen Typen wiederverwendbar sind.

Beispiele zur Typisierung

Die Implementierung einer Funktion `mylen` in Python, Java und Kotlin verdeutlicht Unterschiede in der Typprüfung, Null-Sicherheit und Fehlervermeidung durch das Typsystem.

Fehlerbehandlung in C vs. Rust

- In **C** erfolgt Fehlerbehandlung oft über Rückgabewerte wie `NULL` oder globale Variablen wie `errno`, was fehleranfällig und schwer zu verifizieren ist.
  - Rückgabewert null in malloc zeigt an dass es einen Fehler gab.
  - Negative Rückgabewerte in open zeigen ebenfalls Fehler an.
- **Rust** nutzt den `Result` Typ und Pattern Matching, um Fehler explizit und sicher zu behandeln, wobei der Compiler sicherstellt, dass alle Fehlerfälle berücksichtigt werden.

Speicherverwaltungskonzepte

Memory Safety Issues machen einen großen Teil der Sicherheitsbugs aus, insbesondere in Sprachen wie C.

- **Manuelle vs. automatische Speicherverwaltung**:
  - **C** setzt auf manuelle Speicherfreigabe.
  - Sprachen wie **Java** und **Swift** setzen auf Garbage Collection.

Automatische Speicherverwaltung zur Laufzeit

Vorteile:

- garantierte Speichersicherheit z.B. in Swift und Java.

Nachteile:

- Konstanter Overhead und unerwartetes Pausieren der Ausführung für Garbage Collection.

Automatische Speicherverwaltung zur Übersetzungszeit

Vorteile:

- Kein oder minimaler Performance-Overhead garantierte Speichersicherheit Speicher-Bugs werden zur Übersetzungszeit anstatt zur Laufzeit sichtbar.

Nachteile (speziell in Rust):

- Programmierung ist schwieriger da malloc/free an den beweisbar richtigen Stellen eingefügt werden müssen.

Eigenschaften und Vorteile Rust

Rust implementiert eine automatische Speicherverwaltung zur Übersetzungszeit ohne Performance-Overhead. Konzepte wie Ownership, Moves und Borrowing gewährleisten Speichersicherheit ohne Laufzeitkosten.

Ownership, Moves und Borrowing

- **Ownership**: Jeder Wert hat einen eindeutigen Besitzer, der für die Freigabe verantwortlich ist.
- **Moves**: Übertragung von Ownership bei Zuweisung oder Funktionsaufruf, wodurch frühzeitige Fehlererkennung möglich ist.
  - Primitive Typen (z.B. Integer und Floats) werden kopiert
  - Bei Zuweisung der Ownership überträgt (**Move**).
- **Borrowing**: Referenzen erlauben den sicheren Zugriff auf Daten ohne Ownership-Transfer, verhindern gleichzeitige Schreibzugriffe und Data Races.
  - Variablen in Rust sind entweder immutable oder mutable. Borrowing ist der Akt der Referenzerstellung für einen Wert.

Grenzen des Borrow-Checkers und Safe Abstraktionen

Der Borrow-Checker kann nicht alle sicheren Codes erkennen, insbesondere bei komplexen Strukturen wie doppelt verketteten Listen.

Daher werden `unsafe`-Blöcke und Smart Pointer wie `Rc<T>` und `RefCell<T>` genutzt, um erweiterte, aber dennoch sichere Abstraktionen zu schaffen.

Lesende/Schreibende Referenzen

- ENTWEDER beliebig viele lesende Referenzen

- ODER höchstens eine schreibende Referenz.

Smart Pointer

Smart Pointer sind sichere Abstraktionen z.B. Rc<T> für multiple Ownership und RefCell<T> für mutablen Speicher.

Rust garantiert:

- Speichersicherheit (Verhindern von Memory Leaks)
  - Ein Wert wird genau dann freigegeben wenn der Besitzer den Scope verlässt.
- Verhinderung von Data Races und Dangling Pointers.
  - Referenzen können nicht länger leben als ihr Owner was zu Compile-Fehlern führt wenn versucht wird auf ungültigen Speicher zuzugreifen.
- Fehlerbehandlung wird durch das Typsystem erzwungen sodass Fehler nicht ignoriert werden können
- Zero-Cost Abstractions
  - Das Typsystem und der Borrow-Checker sichern korrekte Speicher- und Fehlerbehandlung bereits zur Kompilierzeit.

Beispiele

Integer-Überlauf in Programmiersprachen

- In **C** führt ein Überlauf zu **undefiniertem Verhalten**, während **Java** das Überlaufverhalten präzise definiert.

Ergebnis des folgenden JavaScript Ausdrucks: `'2' - 3`

Ergebnis: `-1`

Wieso?

In JavaScript ist `-` ausschließlich ein **Arithmetik-Operator**. Daher konvertiert JavaScript den String `'2'` in einen `number` (Type Conversion), bevor die Subtraktion durchgeführt wird.

Was ist das Ergebnis des folgenden PHP Ausdrucks: `'2' + 3`**?**

Ergebnis: `5`

Wieso?

PHP ist eine **schwach-typisierte** Sprache, wie auch JavaScript. Im Gegensatz zu JavaScript versucht PHP bei einer `string + int`-Operation, den String zuerst in einen Integer

umzuwandeln (was hier gelingt).

Was ist das Ergebnis des folgenden Python Ausdrucks: `'5' - 3`**?**

Ergebnis: `TypeError`

Wieso?

Python ist eine **stark-typisierte** Sprache und unterstützt keine impliziten Typumwandlungen. Soll eine Typumwandlung durchgeführt werden, muss dies explizit durch den Entwickler geschehen:

```
(int('5')) - 3
```

Was ist das Resultat von folgendem Rust Code?

```
let x = String::from("esse");
let y = x;
print!("x : {}", x);
```

Ergebnis:

x verlässt den Scope, und ein Compile-Error tritt auf, da x nicht mehr gültig ist.

Wieso?

In Rust wird der Wert von x an y **verschoben** (Move). Dadurch besitzt x keinen Zugriff mehr auf die Daten, und der Compiler verhindert, dass x nach der Übergabe verwendet wird.

Weitere Aspekte bei der Sicherheitsüberprüfung von Programmiersprachen

Sicherheitsprüfung von nicht nur

- Speicher
- Typen

sondern auch

- Kontrollfluss
- Anzahl der Argumente bei Prozeduraufrufen
- spezifische Angriffe wie ROP und Format-String-Angriffe.

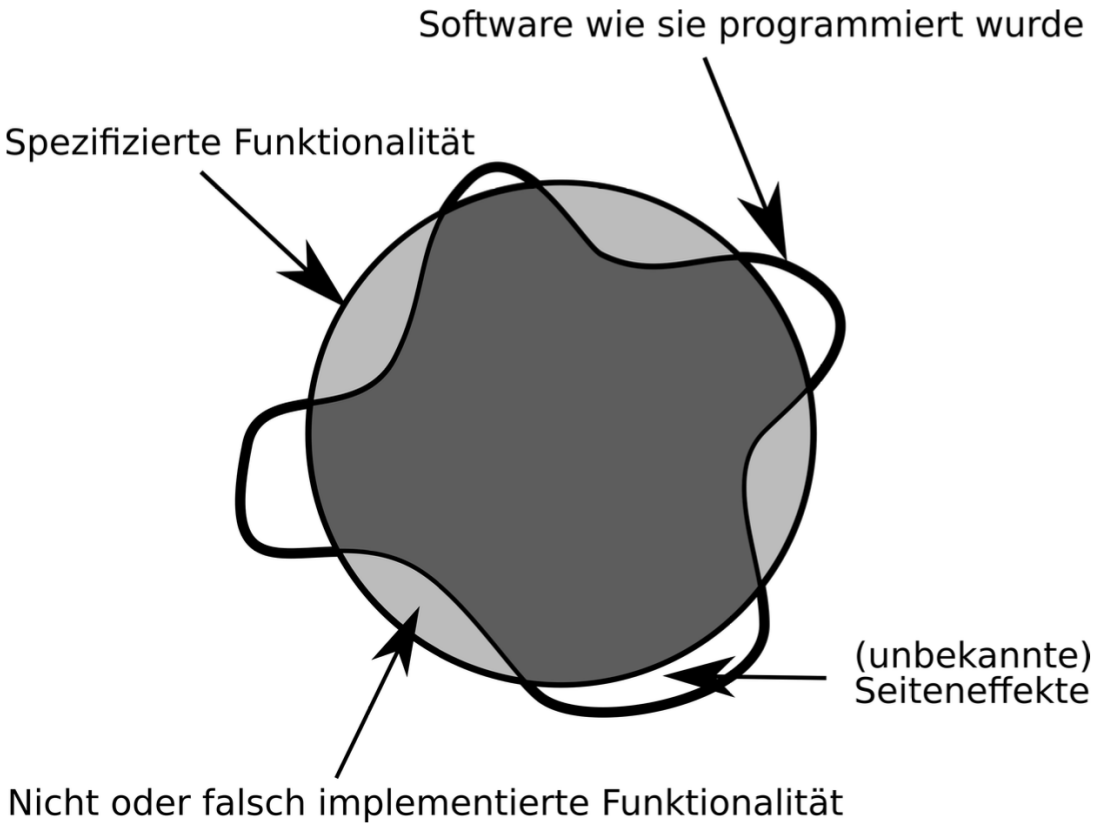
Testing

Testen ist der Prozess, ein Programm oder System durch gezielte Interaktionen in einen definierten Zustand zu versetzen und den aktuellen Zustand mit dem erwarteten zu vergleichen, um festzustellen, ob es sich wie vorgesehen verhält.

- **Validierung**: Erzeugen wir das richtige Produkt?
  - **Vergleich**: Wünsche des Kunden ← → Anforderungen
- **Verifikation**: Erzeugen wir das Produkt richtig?
  - **Vergleich**: Anforderungen ← → Applikation
- **Zielsetzung des Tests**: Die Auffindung von Fehlern.

Korrelation zwischen Testabdeckung und Produktqualität

Je niedriger die Testabdeckung, desto niedriger oftmals auch die Produktqualität, was insbesondere im Bereich der IT-Software oftmals zu hohen Wartungs- oder Bug-Behebungskosten führt.



Wer kann Sicherheitstests durchführen?

Entwickler:in/Tester:in

- Fehlendes Wissen über Siche rheit und aktuelle Schwachstellen!
- Schwachstellen werden leicht übersehen
- Augenmerk auf Funktionalität
- Oft Implikationen aufgrund von detaillierten Wissen über Code
- Testen mit Implikationen kann zu Fehlannahmen führen führen

Sicherheitsexpert:in

- Fehlendes Wissen über Domäne und die Implementierungsdetails -> komplexe Logikflüsse werden leicht übersehen
- Wissen über Schwachstellen und aktuelle Entwicklungen im Sicherheitsbereich, wie zum Beispiel Sicherheits-Testtools
- Testen des Systems aus Angreifersicht ->Realitätsnäher

Merkmalsräume von Software-Tests

Die Merkmalsräume dienen zur Klassifizierung von Tests anhand verschiedener Dimensionen.

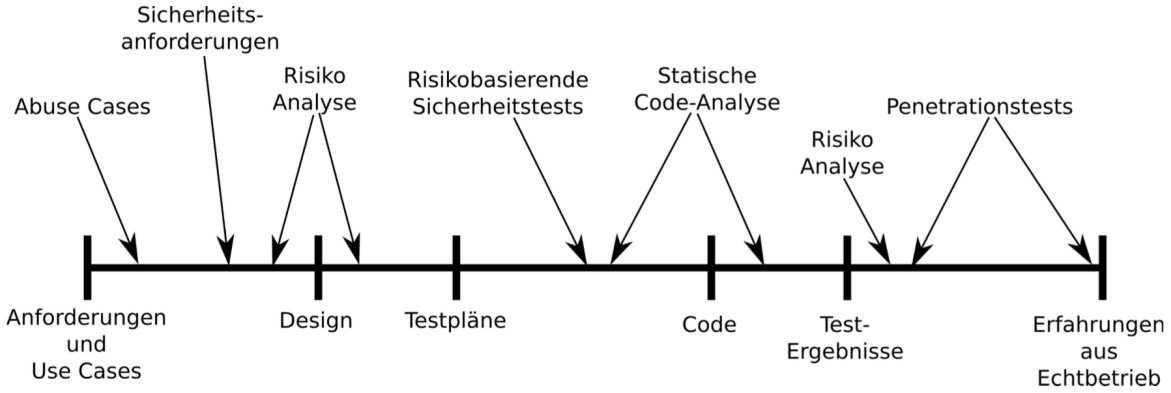
Prüfkriterium

inhaltliche Orientierung des Testfalls (*Was?-Frage*)

- **Funktionale Tests:** Testen der Funktionen (Login, Daten speichern), Spezifikation als Grundlage → Erwarteter Output, bspw. *Login mit gültigen Credentials möglich?*
- **Nicht-funktionale Tests:** Allgemeine Eigenschaften eines Systems (Zuverlässigkeit, Sicherheit, Robustheite gegen Angriffe), bpsw. *Login mit ungültigen Credentials möglich (z.B. SQLI)?*

Prüfebene

Bestimmung des optimalen Zeitpunktes im Lebenszyklus für spezifische Tests (*Wann?-Frage*)



Prüfmethodik

Durchführung des Testfalls (*Wie?-Frage*).

- **White-Box-Tests:** Wissen über Systemdetails.Vergleich tatsächliches Verhalten mit erwartetem Verhalten. → *Vollständige Informationenselten vorhanden*
- **Black-Box-Tests:** Kein Wissen über Systemdetails.Prüfe tatsächliches Verhalten (z.B. durch simulierte Angriffe, Äquivalenzklassen und Grenzwertanalysen), dann Deduktion ob Verhalten problematisch ist. → *Ineffizient aber realitätsnah.*
- **Grey-Box-Tests:** Kombination aus White- und Black-Box-Testing.Datenvarianten anhand interner Details identifizieren (White-Box-Tests) und anschließend Tests mit ermittelten Datenvarianten durchführen (Black-Box-Tests). → *Gängigste Methode in der Praxis*

Testtechniken

Statische Codeanalyse

Prüfung des Codes ohne Ausführung zur Identifikation von Sicherheitslücken (manuelle Reviews wie Secure Code Review und automatisierte Analysen).

Beispiele:

- **Secure Code Review:** manuelle Prüfung von Code auf Sicherheitsbedenken.
  - Top-Down: Wissen über Schwachstellen als Ausgangspunkt. Erfordt kein Wissen über Codedeteils zu Beginn. Review durch Sicherheitsexpert:in.
  - Bottom-Up: Wissen über Code als Ausgangspunkt und schrittweise Identifizierung potenzieller Schachstellen. Review durch Entwickler:in oder Tester:in.
  - Walkthrough: Imaginäres durchdenken von Szenarios am Papier.

    - Inspection: Überprüfung von Codeabschnitten mittels Checklisten bzw. einen Vergleich mit aktuellen Coding Standards.
- Compiler
- automatisierte Scanner
- **Automatisierte statische Codeanalyse**:
    - Prozess
        1. Source Code, Byte Code oder Binaries werden vom Tool eingelesen und in ein repräsentatives Modell umgewandelt.
        2. Anhand einer Datenbank an Sicherheitswissen wird eine Analyse gestartet, z.B. Vorkommen problematischer Anweisungen wie strcpy() in C-Programmen.
        3. Violations werden ausgegeben/geflagt
    - Varianten:
        - **Signaturbasiert**: Suchen nach definierten Mustern wie sprintf, strcpy, ...
        - **Kontrollfluss-Analyse** basierend auf Modell Checking
        - **Datenfluss-Analyse** zur Identifizierung nicht vertrauenswürdiger Quellen und unsicherer Handhabung.

Fuzzing

Einsatz zufälliger oder mutierter Eingaben, um unerwartetes Verhalten und Schwachstellen zu entdecken.

Arten:

- **Unterscheidung nach Input-Erstellung**
    - **Generation-based Fuzzer**: Generierung neuer Daten.
    - **Mutation-based Fuzzer**: Abwandlung bestehender Daten.
- Unterscheidung nach Smartness
    - **Random Fuzzer**: Beispiel: `'HdmxH&k ddadahtrr'`.
    - **Template Fuzzer**: Beispiel: `'GET /ex??ple.html'`.
    - **Block Fuzzer**: Beispiel: `'GET /example.html'`.
    - **Evolution-based Fuzzer**:
        - Lernen von Spezifika des Protokolls durch Output wiederholter Abfragen.

    - Umsetzung eines komplexeren Ablaufs, z. B. Handshake.

Penetration Testing

Realistische Angriffssimulation aus der Perspektive eines Angreifers zur Erkennung von Sicherheitslücken unter realen Bedingungen.

- Die Durchführung simuliert reale Angriffe auf Systeme und berücksichtigt rechtliche und organisatorische Rahmenbedingungen.
- Ein strukturierter Prozess mit Phasen vom Pre-Engagement bis zum Reporting wird vorgestellt.
- Die Notwendigkeit von Kreativität und legalen Vorgaben wird betont, um realistische und sichere Tests zu gewährleisten.

Rechtliche Rahmenbedingungen:

- **Rules-of-Engagement**: Klare Vorgaben, z.B. Vermeidung von Schäden und rechtliche Absicherung.
- **Permission-To-Attack**: Auftraggeber:innen müssen schriftlich bestätigen, dass die Tester:innen das Zielsystem angreifen dürfen.
- Dokumentation von Testschritten, Datenzugriffen und Ergebnissen erforderlich.

Phasen eines Penetrations Tests (PTES-Standard):

1. **Pre-Engagement**: Planung des Tests.
2. **Intelligence** Gathering: Informationsbeschaffung.
3. **Threat Modelling**: Analyse der Bedrohungslage.
4. **Vulnerability Analysis**: Schwachstellenanalyse.
5. **Exploitation**: Ausnutzung der Schwachstellen.
6. **Post-Exploitation**: Bewertung der Auswirkungen.
7. **Reporting**: Erstellung des Berichts.

Ethical Hacking

Ethical Hacking zielt darauf ab, Schwachstellen in Software oder Systemen aufzudecken, um deren Sicherheit zu verbessern.

- **White-Hat Hacker:innen**: Arbeiten **legal** und melden Schwachstellen **verantwortungsvoll**.

- **Gray-Hat Hacker:innen**: Arbeiten teils **ohne Erlaubnis**, veröffentlichen Schwachstellen jedoch nicht für böswillige Zwecke.
- **Black-Hat Hacker:innen**: Missbrauchen Schwachstellen für persönliche Interessen.

Bug Bounty Programme

Bug Bounty Programme sollen Security-Researcher:innen und Hacker:innen unbürokratisch einladen, Systeme auf Schwachstellen zu testen.

- **Belohnungen**: Geld oder andere Anreize für die Meldung realer Schwachstellen.
- **Merkmal**: Kein formaler Penetrationstesting-Prozess, aber ähnliche Ziele.

Vulnerability Disclosure

Vulnerability Disclosure bezeichnet die Veröffentlichung von Schwachstellen, um Hersteller:innen und Nutzer:innen zu informieren.

- **Responsible Disclosure**:
  - Schwachstellen werden zunächst den Hersteller:innen **gemeldet**.
  - Oft wird eine **Frist** an die Hersteller:innen gesetzt. Nach deren Ablauf kann die Frist verlängert werden oder es kommt zu **Full Disclosure**.
- **Full Disclosure**:
  - Alle Details werden direkt **veröffentlicht**.
  - Nutzer:innen haben das Recht, über Bedrohungen informiert zu werden.
  - Hersteller:innen sollen durch öffentlichen Druck zu schnellen Sicherheitsmaßnahmen gezwungen werden.
  - Risiko: Zwischenzeitlicher Missbrauch der Schwachstellen.

Zero-Day Exploit

Ein Exploit wird z. B. auf GitHub **veröffentlicht, ohne Vorwarnung** an den Hersteller.

- In **Responsible Disclosure** bekommt der Hersteller normalerweise eine **Frist** (z. B. 90 Tage), um zu reagieren.
- Daher der Name **0-Day**.

Betriebssystemsicherheit (Linux, Windows)

Aufgaben eines Betriebssystems

Ein Betriebssystem umfasst die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.

Klassifikationen:

- Multiuser, Singleuser
- Singletasking, Multitasking
- Desktop, Mobile
- Real-Time, Embedded

Sicherheitsziele und Maßnahmen

Die Ziele Vertraulichkeit, Integrität und Verfügbarkeit leiten die Analyse der Sicherheitsmaßnahmen ab. Maßnahmen umfassen Berechtigungssysteme, Firewalls, Speicherverwaltung (ASLR, Non Executable Stack), Verschlüsselung, Sandboxing sowie systematische Härtung.

ASLR

Address Space Layout Randomization (ASLR) sorgt dafür, dass Speicheradressen wie Code, Stack und Heap bei jedem Programmstart zufällig angeordnet sind. Das soll Overflow-Angriffe erschweren, da Angreifer für ihre Exploits oft feste Adressen benötigen (z.B. Return-Adresse bei ROP).

ROP

Bei **Return Oriented Programming (ROP)** versucht die Angreiferin bestehende Codefragmente (sogenannte "Gadgets") in einem Zielprogramm zu missbrauchen, um schädlichen Code auszuführen, ohne eigenen Code einzuschleusen.

Gadgets sind kurze Instruktionsfolgen, die typischerweise mit einem ret-Befehl enden und durch gezielte Manipulation des Stack-Pointers aufgerufen werden. ROP umgeht Schutzmechanismen wie den Non-Executable Stack, da es legitimen Code nutzt, und wird oft durch ASLR oder Control-Flow Integrity erschwert.

Non-Executable Stack

Ein **Non-Executable Stack** markiert den Stack-Speicher als nicht ausführbar.

Dies schützt vor Angriffen wie stackbasierten Buffer Overflows, bei denen Angreifer versuchen, schädlichen Code im Stack auszuführen. Angriffe wie Return-Oriented Programming bleiben jedoch möglich.

Systemhärtung

Prinzipien und Herausforderungen

Härten bedeutet die Entfernung unnötiger Softwarebestandteile zur Minimierung der Angriffsfläche. Ideale Härtung erfordert sichere Hardware, minimale Installation und tiefgehende Kenntnisse, was in der Praxis durch Wissens-, Zeit- und Budgetmangel sowie proprietäre Software limitiert wird.

Praktische Härtungsmaßnahmen

Praktische Ansätze reduzieren installierte Pakete, deaktivieren nicht benötigte Dienste, straffen Berechtigungen und ändern Standardpasswörter. Beispiele aus Linux zeigen die Reduktion von Paketen bei CentOS und den Einsatz von Tools wie `find`, `chmod`, ACLs, SELinux und `chroot`.

Package-Manager

Package-Manager installieren oft empfohlene oder erforderliche Abhängigkeiten automatisch. Das vergrößert die Angriffsfläche des Systems, da die installierten Softwarekomponenten oft zusätzliche (unnötige) Funktionen oder potenzielle Fehler enthalten.

Härtungsmaßnahme:

Abhängigkeiten kritisch zu prüfen und nur notwendige Pakete zu installieren/unnötige Pakete deinstallieren bzw. austauschen.

Einschränkung von Berechtigungen Unix/Linux

- User, Group, Others Berechtigungen (ls, chmod, chown, chgrp)
- Access Control Lists (ACLs) für granulare Berechtigungen
- Containerisierung (chroot-jail, ...)
- suid bzw. sgid

Ausnahme: root-user (Ziel vieler Angriffe ist es daher root-Status zu erlangen)

Beispiele root-Berechtigungen

- Dateien/Ordner lesen

- Änderung von Zugriffsberechtigungen
- Wechsel der UID
- Prozesssteuerung
- Verändern von Systemparametern (Limits f. Prozesse, Filesysteme,…)
- User-Management
- System-Management (Shutdown, Reboot,…)
- Aktivierung Promiscous Mode v. Netzwerk Interfaces
- Filesysteme (un)mounten
- chroot

Ressourcen deren Berechtigung im BS verwaltet wird:

- Zugang zur Maschine / IT-System
- Filesystem
- Memory

SUID bzw. SGID

Set User ID (SUID) und **Set Group ID (SGID)** erlauben Programmen, mit den Rechten des Dateibesitzers (z.B. root) oder der Gruppenrechte ausgeführt zu werden.

Problem:

Programme setzen oft unnötig das SUID-Bit (Verletzung von Principle of Least Privilege)

=> Fehlerhafte Programme können von Angreifern (z.B. durch Overflow-Attacken) ausgenutzt werden, um höhere Rechte (z.B. root) zu erlangen.

Härtungsmaßnahme:

Überprüfen, ob alle Programme die SUID setzten es auch wirklich brauchen:

```
sudo find / -perm -4000 -print
```

chroot

Der Befehl **chroot** (change root) ändert das Root-Verzeichnis (`/`) für einen Prozess und dessen Nachkommen. Dies wird oft genutzt, um eine isolierte Umgebung zu schaffen, z.B. für Tests oder eingeschränkte Zugriffsmöglichkeiten (sogenannte chroot-jail).

Probleme:

- Angreifer mit Root-Rechten könnten unter Umständen aus der **chroot-jail** ausbrechen, da `chroot` allein keine umfassende Sicherheit bietet.
- Es ist wichtig, die Anzahl an zusätzlichen Tools innerhalb der `chroot`Umgebung zu minimieren, um potenzielle Angriffsflächen zu verringern.

Zugang zu Bibliotheken:

- **Dynamisch gelinkte Applikationen** benötigen Zugriff auf die entsprechenden Bibliotheken innerhalb des `chroot`Verzeichnisses. Diese Bibliotheken müssen manuell in das `chroot`Verzeichnis kopiert werden.
- **Statisch gelinkte Applikationen** benötigen keine externen Bibliotheken, was die Konfiguration vereinfacht, allerdings größere Binärgrößen zur Folge hat.

Windows-spezifische Aspekte

Firewall (Windows)

Eine **Firewall** ist ein Sicherheitssystem, das den ein- und ausgehenden Datenverkehr überwacht und diesen basierend auf vordefinierten Regeln zulässt oder blockiert.

- **Ziel**: Schutz von Netzwerken oder Endgeräten vor unerwünschtem Zugriff und Angriffen.

Einfaches Paket-Filtering:

- Filtert den Datenverkehr auf Basis statischer Regeln.
- **Kriterien** sind z. B. **IP-Adresse, Portnummer** und **Protokoll**.
- Keine Analyse des Zusammenhangs zwischen Paketen.
- **Beispiel**: Regel erlaubt alle Pakete mit Port 80 (HTTP).

Stateful Inspection (z. B. Windows Firewall):

- Überwacht nicht nur einzelne Pakete, sondern die gesamte Verbindung (den Zustand, z. B. Sitzung).
- **Beispiel**: Erlaubt nur Antworten auf von innen gestartete HTTP-Anfragen.

Programme und Dienste

- **Programme**: Anwendungen, die direkt vom Benutzer gestartet und interaktiv genutzt werden können (z.B. Texteditoren, Browser).

- **Dienste**: Prozesse, die beim Systemstart im Hintergrund gestartet werden und systemweite Funktionen bereitstellen (z.B. Netzwerk, Druckerwarteschlange).

Strategien zur Sicherung von Windows

Mindestens:

- **Start-Verbot:** Starten von Programmen/Diensten standardmäßig verbieten, nur zugelassene Programme erlauben.
- **Allow-Listing**: Ein- und ausgehenden Netzwerkverkehr blockieren, Ausnahmen gezielt definieren.
- **User-Rechte per Default**: Automatische Nutzung von Admin-Rechten deaktivieren.

Weiter:

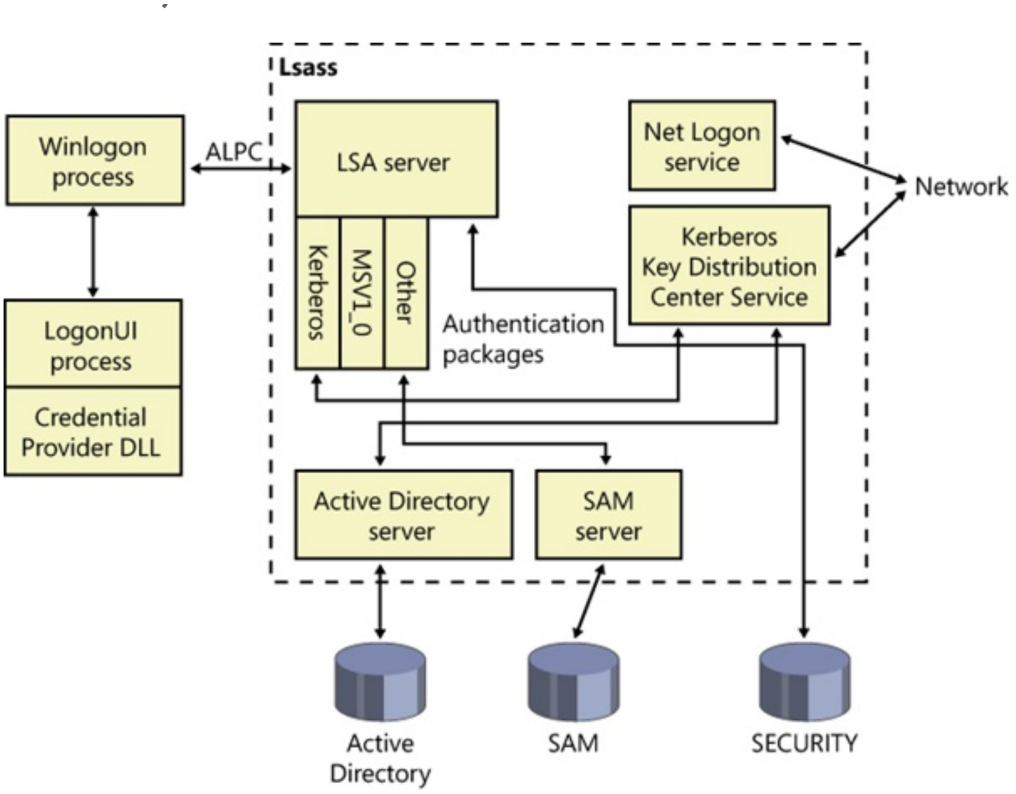
- Kontrollierte Windows-Updates durchführen.
- Software isolieren (z.B. AppContainer).
- Einsatz von Windows Defender Advanced Threat Protection (ATP).

Windows-Login-Prozess

1. **Credentials abfragen**: Benutzeranmeldedaten werden vom **Credential Provider** gesammelt und gehasht.
2. **Weitergabe an LSASS:** Die Anmeldedaten werden an den **LSASS**Prozess (Local Security Authority Subsystem Service) übergeben.
3. **Prüfung und Vergleich:**
    - **LSASS** prüft die Anmeldeinformationen und die Kontorechte.
    - Holt den **Credential-Hash** aus der **SAM-Datenbank** (Security Account Manager) oder aus einem **Active Directory**.
    - Vergleicht den Hash mit den übermittelten Daten.

Elemente im Überblick

- **Credential Provider**: Erfasst Anmeldeinformationen (z.B. Benutzername, Passwort).
- **LSASS**: Führt die Authentifizierung durch und verwaltet Sicherheitsrichtlinien.
- **SAM**: Lokale Datenbank, die Benutzerkonten und zugehörige Hashes speichert => beliebtes Angriffsziel



Mechanismen und Konzepte zur Zugriffssteuerung

- **Login-Prozess**: Authentifizierung durch Prüfung von Benutzer-Credentials.
- **Benutzerkontenverwaltung**: Verwaltung von Benutzern und Gruppen mit spezifischen Berechtigungen.
- **Berechtigungen**: NTFS-, Konte- und Benutzerrechte.
- **Gruppen**: Zusammenfassung von Benutzern mit gleichen Rechten.
- **Windows Access Tokens**: Enthalten Informationen zu Benutzerrechten und Gruppenmitgliedschaften.
- **Kerberos Tickets**: Authentifizierungsmechanismus in internen Netzen analog zu WAT.
- **User Account Control (UAC)**: Trennung von administrativen und normalen Benutzerrechten.

Benutzer:innenverwaltung

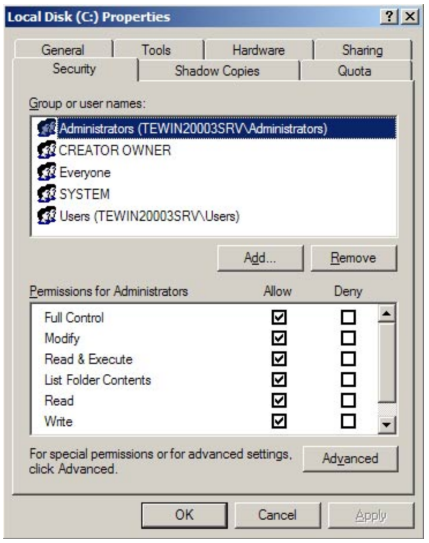
- **Lokale Nutzerverwaltung**: erfolgt über **LSASS** (Verwaltung der Sicherheitsrichtlinien) und **SAM** (Speicherung lokaler Konten).

- **Verzeichnisbasierte Verwaltung** mit **Active Directory** ermöglicht die zentrale Organisation von Nutzer:innen, Gruppen und Computern in Domänen (z.B. Firmennetze). Login durch **AD Controller** über Kerberos-Protokoll -> Austellung von **Kerberos-Ticket**.

NTFS-Rechte

New Technology Filesystem (NTFS) ist das Filesystem unter Windows.

NTFS-Rechte steuern den Zugriff auf einzelne Dateien und Ordner in und ermöglichen eine granulare Rechteverwaltung für Benutzer:innen und Gruppen. Jedes File hat eine Access Control List (ACL).



Kontorechte & Privilegien

- **Kontorechte** beziehen sich auf **Anmelde- und Zugriffsrechte** von Benutzer:innen und Gruppen innerhalb der Computerumgebung, z.B. lokales Anmelden erlauben/verbieten. Sie werden durch **LSASS** überprüft und pro Konto zugewiesen.
- **Privilegien** erlauben bestimmten Nutzer:innen oder Gruppen die Durchführung spezifischer **Systemoperationen**, z.B. das Herunterfahren des Systems (*SeShutdownPrivilege*). Sie werden **durch Admin vergeben** und sind durch **Access Tokens** implementiert. Prozess erben die Access Token des Nutzers.

Gruppen

Gruppen dienen dazu, Berechtigungen und Privilegien zusammenzufassen, um sie an mehreren Benutzerkonten zuzuweisen ("Berechtigungsprofile"). Sie können vom admin erstellt und verwaltet werden.

Standardgruppen

- **Administratoren**: Vollzugriff, Berechtigungen zuweisen.
- **Benutzer**: Standardgruppe mit eingeschränkten Rechten.

- **Gäste**: temporäres Profil, das nach dem Logout gelöscht wird.
- **Hauptbenutzer (Power Users)**: Benutzerkonten erstellen und einfache Systemkonfigurationen vornehmen.

Windows Access Tokens

Ein **Windows Access Token** wird nach der Anmeldung von **LSASS** erstellt und enthält Informationen über Benutzerrechte und Gruppenmitgliedschaften. Jeder vom System erzeugte Prozess erhält eine Kopie des Tokens, um Zugriffsrechte zu prüfen.

Beim Login eines Administrators erstellt LSASS zwei Tokens:

- **Nutzer-Token**: Wird für reguläre Aufgaben verwendet, um das Risiko zu minimieren (Principle of Least Privilige).
- **Admin-Token**: Wird nur bei Aktionen mit erhöhten Rechten genutzt.

Der Wechsel vom Nutzer- zum Admin-Token erfolgt explizit über **User Account Control (UAC)**, um privilegierte Operationen abzusichern.

Kerberos-Tickets

Kerberos Tickets sind das Analog zu Windows Access Tokens, bloß für Windows-Domänen (z. B. Firmennetze).

- Kerberos Tickets werden beim Login vom AD Controller ausgestellt und dienen der Authentifizierung des Nutzers.
- **Aufgaben**: Identitätsnachweis, Zugriffskontrolle auf Domänenressourcen.

Mandatory Integrity Control (MIC) in Windows

Mandatory Integrity Control (MIC) stuft Prozesse nach ihrer Vertrauenswürdigkeit ein und schränkt die maximalen Zugriffsrechte eines Prozesses ein.

- Ziel: Feingranulare Berechtigungskontrolle auf Prozessebene.

Jeder Prozess hat einen Security Identifier (SID), der dessen Integrity Level spezifiziert:

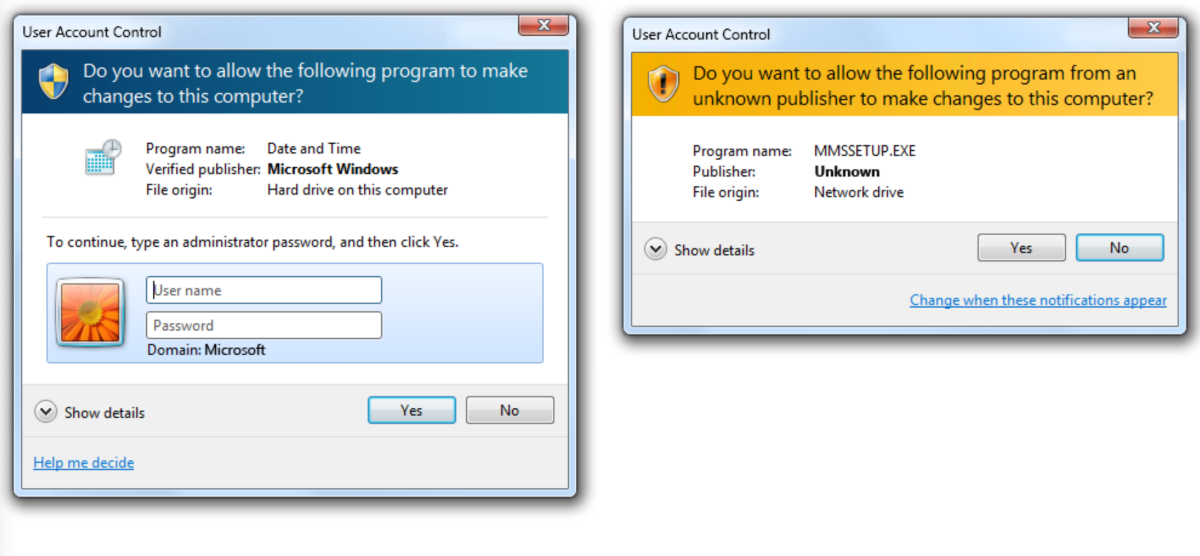
- **System**: Systemprozesse (nur mit Sys-Privileges zugreifbar).
- **High**: Prozesse von Administratoren (nur mit Admin-Privileges).
- **Medium**: Prozesse von Standardnutzern.
- **Low**: Prozesse, die externen Input verarbeiten (z. B. Browser).
- **Untrusted**: Kein Zugriff auf FS, nur Arbeitsspeicher (z. B. Chrome Tabs).

User Account Control (UAC) in Windows

User Account Control (UAC) basiert auf Mandatory Integrity Control (MIC) und Windows Access Tokens und dient dazu, administrative Aktionen abzusichern.

- Standardmäßig werden Prozesse mit **User-Privilegien** ausgeführt, selbst bei Administrator-Konten.
- Für Aufgaben, die erhöhte Rechte erfordern, erscheint ein **UAC-Pop-up**, das entweder nach **Admin-Credentials** fragt (bei User-Konto) oder eine **explizite Bestätigung** verlangt (bei Admin-Konto).

Vorteil: Malware muss bei erfolgreichem Angriff auf Admin-Konto trotzdem zuerst ein Admin-Access-Token erhalten, um UAC zu umgehen.



Gruppenrichtlinien in Windows

Gruppenrichtlinien dienen zur zentralen Konfiguration des Betriebssystems in Windows-Domänen (z. B. Firmennetze) basierend auf der Gruppenzugehörigkeit der Benutzer.

- **Umsetzung durch Group Policy Objects (GPOs):**
  - **Lokale GPOs**: Betreffen genau eine Maschine und erlauben spezifische Konfigurationen.
  - **Globale GPOs**: Betreffen sämtliche angebundene Client-Maschinen und dienen dem Rollout von Standard-Konfigurationen.
- **Einsatzmöglichkeiten:**
  - Durchsetzung von Sicherheitsstandards (z. B. Passwortrichtlinien, Firewall-Basiskonfiguration).

- Überprüfung aktiver Gruppenrichtlinien mit `rsop.msc` oder `gpresult /R`.

Backdoor und Open Source

Eine **Backdoor** ist eine absichtlich eingebaute oder unentdeckte Schwachstelle, die unautorisierten Zugriff auf ein System ermöglicht. Ziel ist oft das Einlesen oder Manipulieren von geschützten Daten.

Ist Open Source eine Lösung?

- Open Source kann Transparenz schaffen, garantiert jedoch nicht die Entdeckung aller Backdoors.
- Vertrauen bleibt ein kritischer Faktor, da viele Open-Source-Bibliotheken von Einzelpersonen verwaltet werden.

Rootkit und Unterschied zur Backdoor

Ein **Rootkit** ist eine spezielle Art von Backdoor, die unautorisierten Zugriff mit **Root-Privilegien** ermöglicht.

- Rootkits sind meist tief ins Betriebssystem integriert und werden direkt beim Systemstart (unentdeckt) aktiv.

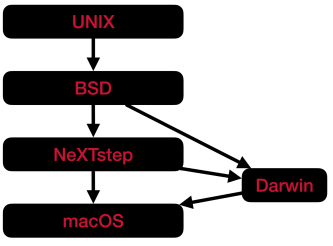
Hinweis:

Eine Backdoor ist ein allgemeiner Begriff für Malware oder Schwachstellen, die unautorisierten Zugriff ermöglichen, jedoch nicht zwangsläufig Root-Privilegien benötigen.

Sicherheitskonzepte von macOS

macOS Architektur und Update-Zyklus

- macOS ist ein UNIX-basiertes Betriebssystem mit Hybrid-Kernel, hauptsächlich in C, C++, Assembler, Objective-C und Swift geschrieben
- Es erscheint jährlich mit neuen Versionen und erhält kontinuierlich Sicherheitsupdates
- Die Unterstützung läuft typischerweise rund sieben Jahre plus zusätzliche drei Jahre für Sicherheitsupdates, wobei die offiziellen Zeiträume unklar bleiben



- Der Closed-Source-Charakter und die exklusive Unterstützung für Apple-Hardware prägen die Architektur, ähnlich einem monolithischen OS mit BSD- und Darwin-Grundlagen.

Sicherheitskomponenten und Dateisystem

Secure Enclave

Die Secure Enclave ist ein **isolierter Prozessor** zur Absicherung sensibler Daten und Verschlüsselungsprozesse.

Sie verwaltet unter anderem FileVault-Verschlüsselung, Biometrie und Schlüssel, isoliert vom Hauptprozessor, um **Side-Channel-Attacken** zu erschweren.

Spezifische Hardware-Komponenten wie TRNG, PKA, AES-Engine und sicherer Speicher gewährleisten zusätzlich hohe Sicherheit und Leistungsfähigkeit.

- **TRNG (True Random Number Generator):** Generiert Zufallszahlen, z. B. für die Erstellung eines Schlüssels.
- **PKA (Public Key Accelerator):** Dedizierter Hardware-Block für Operationen der asymmetrischen Verschlüsselung.
- **AES-Engine:** Zuständig für Operationen der symmetrischen Verschlüsselung.
- **Sicherer nicht flüchtiger Speicher:** Ausschließlich (!) die Secure Element (SE) kann auf diese Komponente zugreifen. Beinhaltet „Entropie“, die zur Generierung von Schlüsseln u. ä. verwendet wird.

Apple File System (APFS) und FileVault

APFS strukturiert Datenträger in Container und Volumes, wobei das System-Volume read-only und signiert ist (SSV) zur Integritätsprüfung.

FileVault nutzt AES-XTS zur Festplattenverschlüsselung auf Volume-Ebene.

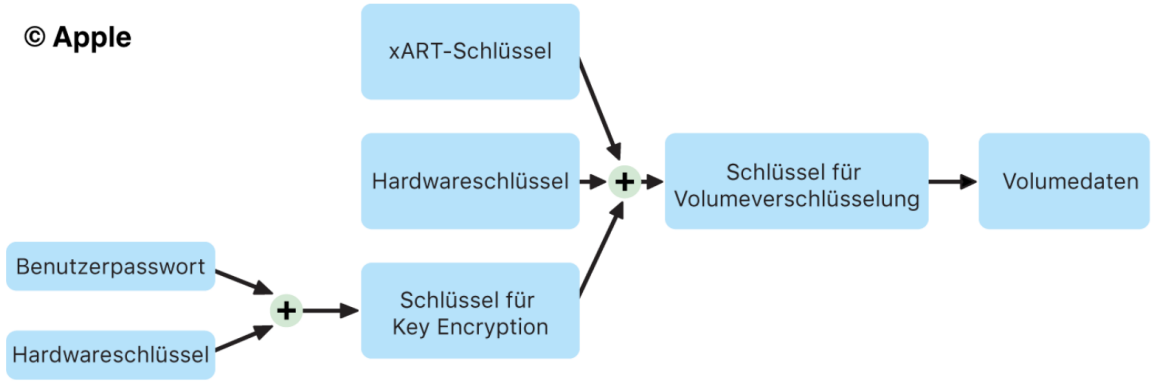
Dabei spielt die Secure Enclave eine zentrale Rolle in der Schlüsselverwaltung.

Es wird zwischen Macs mit und ohne dedizierter Secure Enclave unterschieden, wobei neuere Geräte standardmäßig verschlüsselt sind und die Schlüsselverwaltung isoliert erfolgt.

- **Data Volume**
    - Enthält User:innen-Daten, insbesondere auch von dem:der User:in installierte Programme.

- Geschützt durch **FileVault**.

- **Preboot Volume**
  - Enthält Boot-Informationen.
- **VM Volume**
  - Dient als **Swap File Storage**.
- **Recovery Volume**
  - Enthält **recoveryOS**, inklusive:
    - Festplattendienstprogramm
    - Terminalprogramm



Keychain und Find My

Keychain

- Passwortmanager mit **2FA-Token-Speicher**
- Verwaltet z. B.:
  - Zertifikate
  - Schlüssel
  - Gespeicherte WLANs
- Passwörter etc. können über die Apple ID mit anderen (Apple-)Geräten synchronisiert werden.

Find My

- Wenn ein Gerät verloren geht, kann es über die Apple ID remote gelöscht werden.

- Funktioniert auch offline, indem es über Bluetooth mit nahegelegenen Apple-Geräten kommuniziert.

Private Relay und Intelligent Tracking Prevention

Private Relay verschleiert Benutzeridentität und Surfverhalten, indem es die IP-Adresse und den abgerufenen Inhalt zwischen zwei Relays trennt, ohne zusätzliche VPN-Funktionalitäten. Intelligent Tracking Prevention in Safari blockiert Profiling und Fingerprinting durch on-device Machine Learning und bietet einen Privacy Report.

„Intelligent Tracking Prevention“ in Safari

- Soll **„Profiling“** von Dienstleistenden im Internet verhindern.
- **On-device Machine Learning** soll Tracking-Technologien blockieren.
- Bietet User:innen einen **„Privacy Report“**, in dem blockierte Tracking-Versuche aufgelistet werden.
- Soll ebenfalls gegen **„Fingerprinting“** helfen, indem nur ein vereinfachtes Profil mit Dienstleistenden geteilt wird.

Speicher- und Systemschutz

Technologien wie XD, ASLR, SIP und Secure Boot verhindern Codeausführung, erschweren die Adressierung von Schadcode und schützen kritische Systembereiche. Eingebaute Malware-Erkennung (XProtect), Downgrade-Verbote und Ressourcenzugriffsüberwachung ergänzen den Schutz.

Anwendungsinstallation und Sandboxing

Gatekeeper prüft heruntergeladene Apps auf Signatur und Authentizität. Das Berechtigungssystem von macOS erlaubt feingranulare Zugriffssteuerung auf Hardware-Ressourcen, während Sandboxing isolierten Zugriff von Applikationen gewährleistet.

Mobile Security

Android

Grundlagen und Architektur

Android basiert auf Linux und verwendet den Hardware Abstraction Layer (HAL) zur Standardisierung von Hardwarezugriffen. Es nutzt zusätzliche Sicherheitsmaßnahmen wie SELinux.

Komponenten von Android-Apps

Android-Apps bestehen aus Komponenten wie Activity, Fragment, Service, ContentProvider und BroadcastReceiver.

Activity:

Eine `Activity` ist der "Entry Point" einer Android-App, beinhaltet UI-Komponenten wie Layouts und muss im Manifest deklariert sein.

Content Provider:

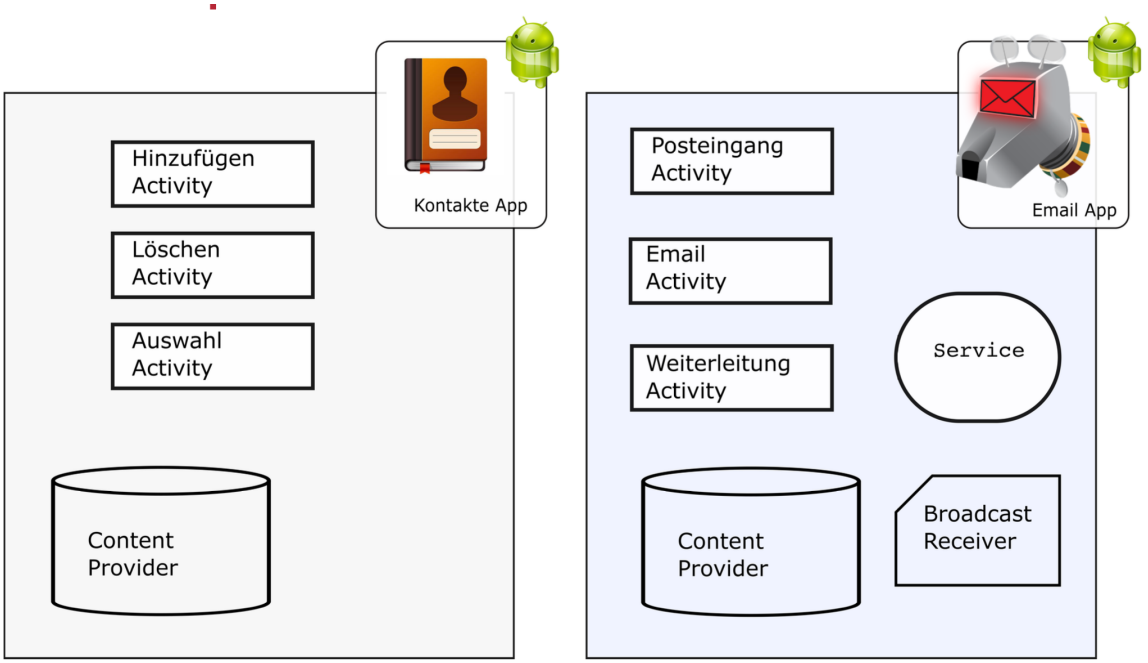
`Content Providers` verwalten den Zugriff auf App-Daten und ermöglichen die gemeinsame Nutzung von Daten zwischen Apps.

APK-Struktur und Bytecode:

Eine APK ist eine ZIP-Datei, die vollständige, installationsbereite Android-Apps enthält.

Der enthaltene Dalvik Bytecode in classes.dex lässt sich durch Tools wie dex2jar dekompilieren.

Zudem erfolgt die Code-Ausführung über die Android Runtime (ART), die seit Android 5 die Dalvik VM ersetzte und durch AOT- sowie JIT-Kompilierung die Performance verbessert.



Sicherheitsaspekte

Das Threat-Modell für Android umfasst verschiedenste Angriffsszenarien, von forensischen Untersuchungen bis zu Netzwerk- oder Codeausführungs-Angriffen.

Das Berechtigungsmodell regelt den Zugriff auf sensible Ressourcen, und die Signierung von APKs sorgt für eine Grundsicherung trotz potenzieller Schwächen bei nachträglichem Code-Nachladen.

Sandboxing sorgt dafür, dass jede App in einer eigenen VM läuft, die durch das Betriebssystem isoliert ist.

App Hardening umfasst Maßnahmen wie Root/Jailbreak-Erkennung und Code-Obfuscation, um Apps sicherer zu machen.

Mögliche Angriffe

- Forensic Attacks
- Network-based Attacks
- Code Execution Attacks
- Web-based Attacks
- Physical Proximity Attacks (USB, NFC)

iOS

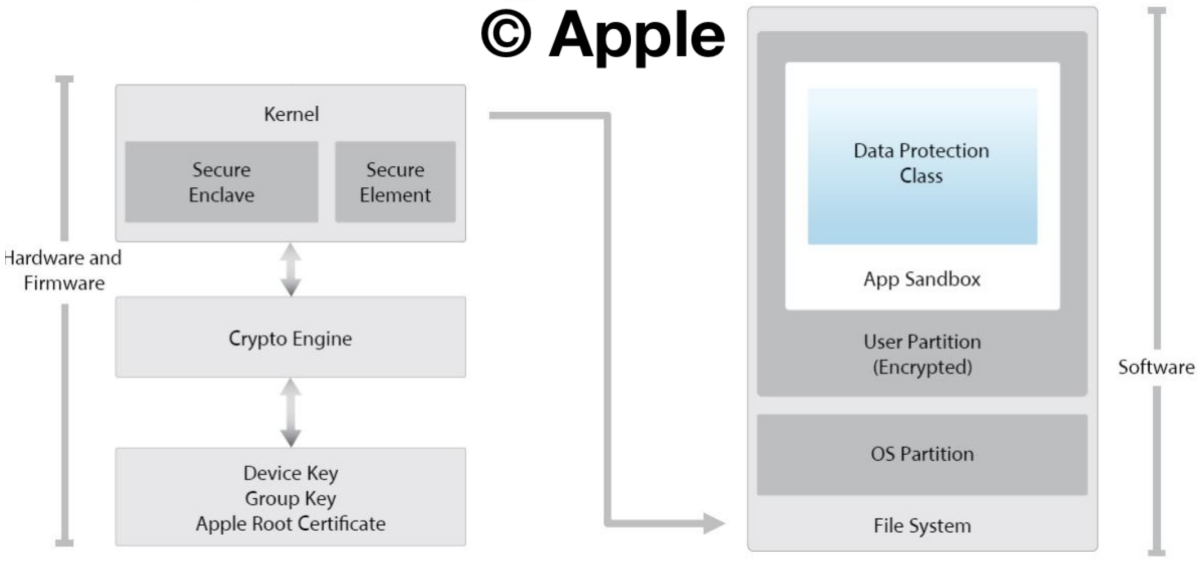
Grundlagen und Architektur

iOS basiert auf macOS/OS X und ist Closed Source.

Die Sicherheitsarchitektur umfasst mehrere Ebenen, angefangen bei der Secure Enclave, einer isolierten Koprozessor-Komponente für kryptografische Operationen, über die dedizierte Crypto Engine bis hin zur Trennung von System- und Benutzer-Datenpartitionen.

Secure Boot:

- jeder Schritt enthält Integritätscheck (kryptographisch signiert von Apple)
- „Chain of Trust“ (Kette des Vertrauens)
- Ziel: Lowest Level Software soll nicht modifiziert werden



Software-Updates und Signierung

iOS-Updates folgen einem strengen Signaturprozess durch Apple, wobei ältere Versionen nach Signaturstopp nicht mehr installiert werden können.

Die Chain of Trust wird vom Boot-ROM bis zum iOS-Kernel umgesetzt. Updates werden entweder als vollständige OS-Kopie oder als differenzielle over-the-air Updates gehandhabt, wobei die Geräte-Exklusiv-ID (ECID) zur Personalisierung und Sicherheit beiträgt.

App-Entwicklung und -Verbreitung

- **iOS Software Development Kit (SDK):** enthält Ressourcen, Technologien & Werkzeuge für iOS-App-Entwicklung – kostenlos
- Entwicklung erfolgt ausschließlich auf macOS mit Xcode und bevorzugt in Swift (früher Objective-C).
- **Vertrieb:**
  - App Store
  - strengere Prüfungen hinsichtlich Sicherheit und Datenschutz vorausgesetzt werden.
- **Codesignierung:** stellt sicher, dass nur autorisierte und unveränderte Anwendungen auf dem System ausgeführt werden.
- **Sandboxing:** trennt Apps voneinander, sodass sie nicht auf Dateien oder Ressourcen anderer Apps zugreifen können.

Aufbau und Sicherheit von iOS-Apps

- **iOS App Store Package (IPA):** enthält eine signierte App-Bundle-Struktur
- Jede App läuft in einer Sandbox, die den Zugriff auf Daten und Ressourcen strikt einschränkt.
- Schlüsselkomponenten sind unter anderem die Bundle-ID, Verzeichnisstrukturen für Apps und Daten sowie die Info.plist-Datei, die Konfigurations- und Berechtigungsanfragen regelt.
- Weitere Sicherheitsmechanismen wie die Keychain zum Schutz sensibler Daten und Frameworks wie SQLite und Core Data zur lokalen Datenspeicherung

Weitere Sicherheitsaspekte

- Caching-Methoden
- automatische Screenshot-Erstellung
- User Defaults

Web Application Security

Technische Grundlagen moderner Web-Applikationen

Komponenten

Webserver

- **Statische Web-Server**
  - Anzeigen von statischen Inhalten (ursprüngliche Idee des WWW).
- **Dynamische Web-Server**
  - Bereitstellung von statischen und dynamischen Inhalten.
  - Verschiedene Web-Applikationen können auf einem Server betrieben werden.
- **Reverse-Proxy**
  - Serviciert oft mehrere Web-Server.
  - Befindet sich zwischen Client und Web-Server.
- **Beispiele für Web-Server**
  - Apache HTTP Server
  - Apache Tomcat

    - Microsoft IIS
    - lighttpd
    - nginx

Weiter Komponenten:

- URLs
- HTTP
- HTML
- CSS
- JavaScript

HTTP Request Methoden:

- GET (Daten abrufen)
- POST (Daten senden)
- PUT (Daten hochladen)
- DELETE (Daten löschen)
- OPTIONS (unterstützte Methoden abfragen)
- HEAD (nur Header zurückgeben)
- TRACE (Änderungen nachvollziehen).

Architektur und Protokolle

Webapplikationen nutzen eine Vielzahl von Architekturen und Protokollen, um den Anforderungen von Interaktivität und Sicherheit gerecht zu werden. Insbesondere HTTP und TLS sind zentrale Bestandteile moderner Webtechnologien.

Session Management und Authentifizierungsverfahren

HTTP ist stateless, daher wird Session Management mittels Cookies, Hidden Fields, URL-Parametern, Header Tokens oder JSON Web Tokens (JWT) realisiert.

HTTP Basic (Basic Auth)

- Authentifizierung erfolgt über **Benutzer:innenname** und **Passwort**, getrennt durch `":"`.
- **Base64-Encoding** wird zur Übertragung der Credentials verwendet.

- Beispiel:
  - **Server:** `WWW-Authenticate: Basic realm="ESSE"`
  - **Client:** `Authorization: Basic c2VjdXJlLXVzZXI6ZXNzZQ==`
  - Decodiert: `secure-user:esse`
- **Sicherheitsrisiko:** Credentials werden nahezu im **Klartext** übertragen.

Digest Auth:

- **Digest Auth** verwendet MD5 zur Verarbeitung der Credentials.
- Ablauf:
  - **Server:** Fordert Authentifizierung mit `WWW-Authenticate` Header an.
    - Beispiel:

      `WWW-Authenticate: Digest realm="ESSE", nonce="...", algorithm=MD5, domain="/rest/", qop="auth"`
  - **Client:** Sendet Authentifizierungsdaten im `Authorization` Header.
    - Beispiel:

      `Authorization: Digest username="secure-user", realm="ESSE", nonce="...", uri="/rest/", algorithm=MD5, response="...", qop=auth, nc=00000001, cnonce="..."`
- Vorteile:
  - Credentials werden nicht direkt übertragen.
  - Authentifizierung erfolgt durch einen MD5-Hash basierend auf:
    - Benutzer:innenname, Passwort, Server-Nonce, Client-Nonce und Request-Daten.

Secure Cookie

- Ein **Secure Cookie** darf nur über **TLS-verschlüsselte Verbindungen** übertragen werden.
- Die Umsetzung dieser Einschränkung liegt in der Verantwortung des **Clients** (Browser), nicht des **Servers**.

JSON Web Token (JWT)

JSON Web Tokens sind eine moderne Methode des Session-Handlings

Aufbau:

- **Header:** Enthält Metadaten, z. B. `"typ":"JWT"` und `"alg":"HS256"`.
- **Payload:** Beinhaltet Claims (vordefinierte oder benutzerdefinierte Informationen), z. B.:

```
{"iss":"joe", "exp":1551984189, "<http://example.invalid/is_root>":true}
```

- **Signatur:** Generiert mit dem im Header definierten Algorithmus, z. B.:

```
HMACSHA256(BASE64URL(UTF8(Header)) + '.' + BASE64URL(UTF8(Payload)), secret-key)
```

Verwendung:

- JWTs können:
  - In **Cookies** gespeichert werden.
  - Als **Authorization-Header** bei HTTP verwendet werden.

Typische Sicherheitslücken und Gegenmaßnahmen

Injection-Angriffe

- Entstehen durch manipulierte Eingaben, die von Interpretern falsch verarbeitet werden.
- Typen: SQL, Command, Template, XML, XPath.
- Gegenmaßnahmen:
  - Strikte Eingabevalidierung.
  - Verwendung von Prepared Statements.
  - Sichere Template-Engines.
  - Eingeschränkte Benutzerrechte.

Kryptographische Fehler und Transport Layer Protection

- Fehler wie schwache Algorithmen oder unzureichender Transportschutz (fehlendes TLS) gefährden Daten.

- Maßnahmen:
  - Verwendung aktueller Verschlüsselungsstandards.
  - Schlüsselmanagement.
  - Getrennte Speicherung von Schlüsseln und Daten.
  - HTTP Strict Transport Security (HSTS).

Broken Authentication und Access Control

- Schwachstellen führen zu Session-Hijacking und unberechtigtem Zugriff.
- Maßnahmen:
  - Überprüfung von URL-Zugriffsrechten.
  - Sichere Generierung von Session-IDs.
  - Regelmäßige Logout-Mechanismen.
  - Strenge Backend-Validierung.
  - Rollenbasierte Zugriffsmodelle.
  - Vermeidung direkter Benutzer-URL-Referenzen.

Security Misconfiguration und Verwendung verwundbarer Komponenten

- Fehlerhafte Konfigurationen und der Einsatz von Komponenten mit bekannten Schwachstellen bieten Angreifern Chancen.
- Maßnahmen:
  - Regelmäßige Audits.
  - Hardening-Prozesse.
  - Zeitnahe Updates.

Local File Inclusion und Google Hacking

- Local File Inclusion ermöglicht unbefugte Dateizugriffe auf dem Server.
- Google Hacking nutzt Suchmaschinen, um verwundbare Webanwendungen aufzuspüren.
- Gegenmaßnahmen:

- Regelmäßige Sicherheitsprüfungen.

Identification and Authentication Failures / Broken Authentication and Session Management

- HTTP ist ein **stateless Protokoll**.
- **Session-Informationen** werden verwendet, um Statusinformationen zu speichern.

Gefahren:

- **Session-Hijacking:** z.B. Tools wie **Firesheep** können verwendet werden, um aktive Sessions zu übernehmen.
- **Übernahme von Benutzer:innen-Accounts**
- **Unsichere Session-Verwaltung: z.B.** Verwendung von **unzureichend zufälligen Session-IDs**.

Maßnahmen:

- TLS verwenden
- Zufällige Session-IDs
- Logout-Mechanismen

Insecure Direct Object References (Broken Access Control)

- Ähnlich zu **Failure to Restrict URL Access**.
- Direkter Zugriff auf Objekte über **vorhersehbare IDs**.
- **Typische Fehler:**
  - Direkter Zugriff auf Objekte, z. B.:
    - `www.insecure-server.invalid?account=100`
    - `www.insecure-server.invalid?account=102`
  - Nur bestimmte Objekte werden im Presentation-Layer versteckt, aber nicht serverseitig überprüft.
  - Keine Kontrolle oder Validierung auf **Server-Seite**.

Gefahren:

- Angreifer:innen probieren ähnliche IDs oder Referenzen aus, um unberechtigten Zugriff zu erlangen.

Maßnahmen:

- **Keine direkten Referenzen verwenden:**
  - IDs durch **zufällige, temporäre Mapping-Werte** ersetzen, z. B.:
    - Statt `?account=100` → `?account=8w9e9fgi`.
    - Statt `?file=accounting.xls` → `?file=119347`.
- **Serverseitige Validierung:**
  - Überprüfen der Objekt-Referenzen auf:
    - **Formatierung**.
    - Ob die Benutzer:innen die **Berechtigung** zum Zugriff haben.

Unvalidated Redirects and Forwards

- Web-Applikationen leiten häufig Benutzer:innen zu anderen Webseiten weiter.
- Die **Zieladresse** wird oft über Benutzer:inneneingaben bestimmt.

Gefahren:

- Angreifer:innen können Benutzer:innen auf **Malware- oder Phishing-Seiten** umleiten.
  - Beispiel: `http://example.invalid/redirect.php?url=http://evil.invalid`
- **Umgehung von Autorisierungsmechanismen** durch manipulierte Weiterleitungen.

Maßnahmen:

- **Keine URLs aus Benutzer:innen-Parametern** für Redirects verwenden.
- **Eingabevalidierung**, um sicherzustellen, dass nur erlaubte Zieladressen akzeptiert werden.

Security Misconfiguration

Die Sicherheit von Web-Applikationen hängt auch von darunterliegenden **Software-Schichten** ab (z. B. aktuelle Patches, Services, Ports).

Gefahren:

- **Ausnutzung von Sicherheitslücken**, da keine Patches installiert sind.
- **Default-Accounts** und Standard-Passwörter bleiben aktiv.
- Angriffe wie:

    - **Auslesen und Verändern von Daten**.
    - **Denial of Service (DoS)**.
- Fehlende **Härtungen** der Systeme.
- Kein Senden von **Security-Headern**.

Maßnahmen:

- **Hardening-Prozesse** definieren und umsetzen.
- Regelmäßiges **Scanning und Auditing** der Konfigurationen.
- Verwendung eines **sicheren Softwaredesigns**.

Vulnerable and Outdated Components

Sicherheitslücken treten regelmäßig in unterschiedlichster Software auf, auch in Komponenten, die im Web verwendet werden.

Gefahren:

- **Bekannte Sicherheitslücken** können leicht ausgenutzt werden, da sie oft **frei verfügbar** dokumentiert sind.
- Diese Lücken ermöglichen **verschiedenste Angriffe**.

Maßnahmen:

- Berücksichtigung von **Security im gesamten Lebenszyklus**:
    - **Analyse, Design, Implementierung, Test, Betrieb**.
- **Zeitnahe Installation von Patches**, um Sicherheitslücken zu schließen.

Local File Inclusion (LFI)

- **File Inclusion (FI)** ermöglicht Angreifer:innen, Dateien in eine Anwendung einzubinden.
- LFI bezieht sich auf die Einbindung von **bereits auf dem Server vorhandenen Dateien**.
- Nutzt oft einen in der Zielanwendung implementierten Mechanismus zur **dynamischen Dateieinbindung**.
- Ursache ist häufig **mangelhafte Input-Validierung**.

Gefahren:

- Anzeigen des Inhalts von Dateien.

- Codeausführung auf:
  - Webserver-Seite.
  - Client-Seite, z. B. durch JavaScript.
- Denial of Service (DoS).
- Offenlegung sensibler Informationen.

Browser Security und abschließende Zusammenfassung

- Browser und Plugins stellen Angriffspunkte dar.
- Schutzmechanismen:
  - Same-Origin Policy.
  - Cookie-Security.
  - Port-Blocking.
  - Sandboxen.
- Herausforderungen durch die Komplexität moderner Browser.

Social Engineering und IT-Sicherheit

- **Social Engineering** nutzt **psychologische Manipulation**, um Sicherheitsmechanismen zu umgehen.
- Es ist die **Kunst, Menschen zu beeinflussen und zu lenken**, oft durch Täuschung, Lügen oder betrügerisches Auftreten.
- Ziel: **Unberechtigter Zugang zu Informationen**, nicht Zerstörung.

Der Social Engineer

- **Eigenschaften:**
  - Hohe soziale Kompetenz, Überzeugungskraft und Geduld.
  - Anpassungsfähig und geschickt in der Informationsbeschaffung.
  - Persönlichkeitstyp „**Neophiliac**“: Ablehnung von Tradition, Streben nach Neuem, schnelles Lernen.
- **Strategien:**

- Baut Vertrauen auf, manipuliert Opfer gezielt und nutzt deren Verhalten aus.

Menschliche Faktoren von Social Engineering

- **Reziprozität:** Menschen fühlen sich verpflichtet, Gefälligkeiten zu erwidern.
- **Konsistenz:** Opfer werden in eine Serie von Entscheidungen gelenkt, um sie zu größeren Aktionen zu bewegen.
- **Soziale Bewährtheit:** Orientierung an anderen in unsicheren Situationen.
- **Sympathie:** Sympathische Personen erhalten schneller Hilfe oder Zustimmung.
- **Autorität:** Entscheidungen von Autoritätspersonen werden selten hinterfragt.
- **Knappheit:** Seltene Ressourcen oder begrenzte Zeit üben Druck aus und fördern impulsives Verhalten.

Phasen eines Social Engineering Angriffs

1. **Informationsbeschaffung:** Recherche von Daten über das Ziel.
2. **Beziehungsaufbau:** Aufbau von Vertrauen durch Identitätsannahme.
3. **Beziehungsausnutzung:** Manipulation zur Informationsgewinnung.
4. **Ausführung:** Finale Angriffsphase, z. B. Datenklau oder Systemzugriff.

Arten von Social Engineering

- **Technology-based Social Engineering:**
  - Z. B. Phishing: Manipulierte E-Mails oder Webseiten zur Datenerhebung.
- **Human-based Social Engineering:**
  - Direkte Kommunikation, oft über Telefon, kombiniert mit Identitätsdiebstahl.
- **Reverse Social Engineering:**
  - Angreifer erschafft ein Problem, bietet Hilfe an und gewinnt so Vertrauen.

Schutzmaßnahmen

- **Traditionelle Maßnahmen:**
  - Schulungen zur Sensibilisierung von Mitarbeiter:innen.
  - Begrenzte Wirksamkeit, da menschliches Verhalten schwer langfristig zu ändern ist.

- **Moderne Ansätze:**
  - Mehrschichtiges Verteidigungsmodell:
    - **Äußere Schicht:** Begrenzung der Informationsverfügbarkeit.
    - **Innere Schicht:** Schulung und Bewusstseinsbildung.
    - **Technische Schicht:** Systemische Maßnahmen wie Spamfilter.
- **Systemische Sicherheitsmodelle:** Kombination organisatorischer, technischer und menschlicher Maßnahmen.

Bewertung des Sicherheitsniveaus

- **Risiko vs. Sicherheit:**
  - Absolute Sicherheit ist unmöglich.
  - Sicherheit erhöht Kosten, erfordert jedoch ein optimales Kosten-Nutzen-Verhältnis.
- **Menschliche Schwächen:**
  - Selbstüberschätzung kann zu falschen Entscheidungen führen.
  - Experten sind oft anfälliger für Betrug in ihrem Fachgebiet, da sie Warnzeichen ignorieren.
- **Bewusstseinsbildung:** Sensibilisierung und Training können die Wahrscheinlichkeit erfolgreicher Angriffe reduzieren.

Usable Security und Trust

Vertrauen und IT-Sicherheit

- Vertrauen ist ein **risikobehafteter Prozess**, bei dem eine Entität sich verwundbar macht, in der Hoffnung auf wohlwollendes Handeln.
- **Mistrust vs. Distrust:**
  - **Mistrust:** Fehlendes Vertrauen, das auf Unsicherheit basiert.
  - **Distrust:** Aktives Misstrauen aufgrund negativer Erfahrungen.
- Vertrauen und Misstrauen sind **keine binären Konzepte**, sondern bewegen sich auf einem **Kontinuum**, was Auswirkungen auf Entscheidungen in der IT-Sicherheit hat.

Folgen und Herausforderungen des Vertrauens

- **Unangebrachtes Vertrauen** birgt Risiken:
  - **Datenverlust**, **finanzielle Schäden**, **ethische Konflikte**.
- **Übermäßiges Misstrauen** behindert die Nutzung sicherer Technologien:
  - Z. B. Verzicht auf Passwortmanager oder biometrische Verfahren.
- **Balance zwischen Sicherheit und Nutzbarkeit:**
  - Vollständiges Misstrauen gefährdet Vertrauen in Open-Source-Software, Compiler und Hardware.
  - Technologieeinsatz und Sicherheitsbedenken stellen immer einen **Trade-off** dar.

Faktoren, die Vertrauen beeinflussen

- Vertrauen in Technologie wird durch folgende Faktoren geprägt:
  - **Vertrauenshaltung.**
  - **Reputation.**
  - **Kosten-Nutzen-Abwägung.**
  - **Usability.**
  - **Technologische und organisatorische Sicherheit.**
- Diese Faktoren sind individuell unterschiedlich gewichtet und komplex miteinander verknüpft.

Usable Security

- **Usable Security** ist ein **interdisziplinärer Ansatz**, der Sicherheit und Benutzerfreundlichkeit vereint.
- Drei Perspektiven:
  - **Sicherheitszentriert.**
  - **Nutzerzentriert.**
  - **Balance zwischen beiden.**
- Ziel: **Sicherheitsmechanismen intuitiv nutzbar** machen, sodass sie nicht als hinderlich wahrgenommen werden.

- Ein schlechtes Interface kann die Fähigkeit der Nutzer:innen, Aufgaben sicher zu erledigen, stark beeinträchtigen.