IT-Sicherheit

Kapitel 5: Applikationssicherheit Teil 2

- Fehlerbehandlung
- Sicherheitstests
- Logging
- Sicherer Betrieb
- Cloud Security
- IoT Security

Prof. Dr. Reiner Hüttl, FH Rosenheim, © 2021, 15.03.2021

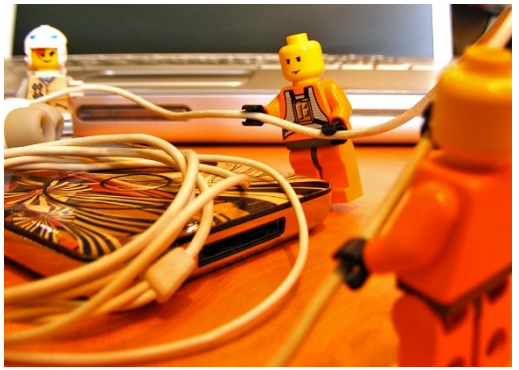
Session Management

- Session Management ist die Grundlage zur Prüfung von Authentisierung und Zugriffskontrolle in zustandsbehafteten Anwendungen

- **Session-Hijacking**
  - Angreifer hat Zugriff auf die Session ID eines Opfers und umgeht damit die Authentifizierung

- **OWASP Top 2: Broken Authenication**
  - Anwendungsfunktionen, die im Zusammenhang mit Authentifizierung und Session-Management stehen, werden häufig fehlerhaft implementiert
  - Dies erlaubt es Angreifern Passwörter oder Session-Token zu kompromittieren
  - Damit können sie die Identität anderer Benutzer vorübergehend oder dauerhaft annehmen

Maßnahmen gegen Session-Hijacking



- **Geheimhaltung der Session ID** (primäre Maßnahme !!)
- **Übertragungskanal sichern** (verschlüsseln, z.B. Https)
- Lebenszeit der Session begrenzen (**Time out**)
- Eine neue **zufällige Session-ID** generieren wenn Benutzer sich anmeldet
- Aus einer bekannten Session-ID dürfen keine fremden Session-IDs rekonstruierbar sein
- Keine persistente Sessions verwenden
- Keine parallelen Session eines Benutzers zulassen (no concurrent sessions)
- **Session Binding**: Session-ID an die IP-Adresse vom Client oder den Browser Fingerprint binden
- Inhalt von Authentifizierungs-Cookies verschlüsseln
- Cookie Attribute setzen (secure, HttpOnly, domain, expires)
- Anti-CSRF-Token (Synchronizer Token bietet Schutz vor Replay-Angriffen)
- **Logout** Option einbauen

Weitere Bedrohungen für Webanwendungen

- **Cross-Origin-Zugriffe**
  - Problem: Microservices, REST APIs wollen Cross-Domain Zugriffe
  - Maßnahme: **SOP Same Origin Policy**
  - **CORS (Cross Origin Resource Sharing):** Zugriff für JavaScript auf andere Origins über http-response Header gesteuert

- **Clickjacking**
  - Angreifer überlagert Webseite mit iFrames und verleitet Benutzer unsichtbare Aktionen auszuführen
  - Gegenmaßnahme: **Frame-Busting** verhindert Darstellung in Frames
    (X-Frame-Options: DENY/ SAMEORIGIN setzen)
  - **CSP Content Security Policy**
    z.B. JavaScript nicht inline sondern nur in Dateien erlaubt, Content nur type text/json
    (aktuell ist CSP W3C Working Draft, nicht von allen Browsern unterstützt)
    https://content-security-policy.com

Weitere Bedrohungen für Webanwendungen

- **Direct object reference**
  - Zugriff auf Objekte über Modifikation der URL (z.B. invoice-ID ändern)
  - Maßnahmen: Access Control vor Zugriff, keine direkten Objekt-Referenzen in URL, indirekte Objekt Referenzen
- **Path Traversal**
  - Zugriff auf Dateien des Webserver
  - Ursache: unzureichende Validierung
  - Maßnahmen: Indirekter Zugriff auf Dateiname, Whitelist-Validierung, Normalisierung
- **Replay-Attacken**
  - Wiederholung von Request durch Angreifer oder versehentlich durch Benutzer
  - Maßnahme: Anti-Replay Token, Anti-CSRF-Token

Weitere Bedrohungen für Webanwendungen

- **Elevation of Privileges**
  - Schwachstellen, die es einem Angreifer ermöglichen, seine Privilegien zu erweitern
  - **Horizontal**: Zugriff auf Ressourcen anderer Benutzer derselben Berechtigungsstufe
  - **Vertikal**: Zugriff auf Ressourcen einer höheren Berechtigungsstufe (z.B. Admin)
- **Remote Code Execution (RCE)**
  - Angreifer kann über eine Schwachstelle Code auf den Zielmaschinen laufen lassen
  - Z.B. Command Injection, Path Traversal und ausführen von Dateien, Ausnutzen von Schwachstellen im Netzwerk oder den IT-Systemen
  - RCE ist absoluter Worst case!
  - Maßnahmen: Defense in Depth, Updating all Components and libraries

DOS Denial of Service

- Kann nicht verhindert werden
- Erstellung eines Notfallplans, damit im Falle eines DoS schnell reagiert werden kann
- Verschiedene Varianten: Anwendungsabsturz, OS-Absturz, CPU-, Speicher- oder Ressourcenüberlastung
- Maßnahmen um Anfälligkeit zu reduzieren
  - Soliden Code schreiben
  - Gründliche Tests und Leistungsanalysen mit Profiler
  - Keinen Daten übers Netzwerk trauen
  - Aufwendige Operationen nur zulassen wenn man sicher ist das ein vertrauenswürdiger Client am anderen Ende sitzt
  - Anwendung verändert Verhalten wenn Angriff erfolgt, z.B. anwachsende Zeitlimits, Clients ohne Authentifizierung abtrennen
  - Ressourcen so spät wie möglich anfordern und so früh wie möglich freigeben
  - Anfragen validieren vor Absenden von Fehlermeldungen,
    keine unnötigen Fehlermeldungen, kontrollierte Fehlerbehandlung

Behandlung Fehlerhafter Eingaben

- Eine saubere und stabile Fehlerbehandlung ist wichtig für sichere Software
  - Sichere Software kann langsamer werden, aber sie sollte nicht abstürzen oder unkorrekte Ergebnisse liefern.
  - Fehler können dazu führen, dass der Benutzer mehr Rechte bekommt als geplant
  - Nach einem Fehler muss eine Anwendung in einen stabilen und sauberen Zustand gebracht werden.
- Nie eine ungültige Eingabe korrigieren um eine gültige daraus zu machen
- Protokollierung der Fehler (Logging)
  - Die System-Logs (z.B. vom Web-Server) sind meist nicht ausreichend
  - Zusätzlich Logging auf Anwendungsebene

Regeln zur Ausnahmebehandlung

- Falls man einen sicherheitsrelevanten Fehler findet
  - Fehler korrigieren und in anderen Code-Teilen nach ähnlichen Problemen suchen
  - Korrektur so nahe wie möglich an der Schwachstelle
  - Behandlung der Ursache nicht der Symptome
  - Die Testtreiber um den Fehler erweitern
- Keine ausführlichen Fehlermeldungen an den Client senden
  - Z.B Datenbankfehler, Zugriffsverletzungen, Dateinamen, Tabellennamen
  - Angreifer bekommt Informationen zu internen Strukturen der Anwendung
  - Alle Fehler abfangen und nur allgemeine Fehlermeldungs-Seite anzeigen
  - Detaillierte Fehlerbeschreibung in Log-Datei
  - Log-Dateien auf Anwendungsebene erzeugen

Information concealment is the first line of defense

Anti Pattern: Fehlerausgabe an den Benutzer

Information disclosure and leakage

HTTP Status 500 - An exception occurred processing JSP page /basket.jsp at line 244

type Exception report

message An exception occurred processing JSP page /basket.jsp at line 244

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
org.apache.jasper.JasperException: An exception occurred processing JSP page /basket.jsp at line 244

241:                                    stmt.execute();
242:                                    stmt.close();
243:                            } else {
244:                                    stmt = conn.prepareStatement("UPDATE BasketContents SET quantity = " + Integer.parseInt(value) + " WHERE basketid=" + basketId +
245:                                                    " AND productid = " + prodId);
246:                                    stmt.execute();
247:                                    if (Integer.parseInt(value) < 0) {


Stacktrace:
        org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:568)
        org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:455)
        org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:395)
        org.apache.jasper.servlet.JspServlet.service(JspServlet.java:339)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:727)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
```

root cause

```
javax.servlet.ServletException: java.sql.SQLException: Unexpected token: % in statement [UPDATE BasketContents SET quantity = 4 WHERE basketid=%27 AND productid = 20]
        org.apache.jasper.runtime.PageContextImpl.doHandlePageException(PageContextImpl.java:916)
        org.apache.jasper.runtime.PageContextImpl.handlePageException(PageContextImpl.java:845)
        org.apache.jsp.basket_jsp._jspService(basket_jsp.java:390)
        org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:727)
        org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:432)
```

note The full stack trace of the root cause is available in the Apache Tomcat/7.0.2 logs.

Apache Tomcat/7.0.2


```
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
```

note The full stack trace of the root cause is available in the Apache Tomcat/7.0.2 logs.

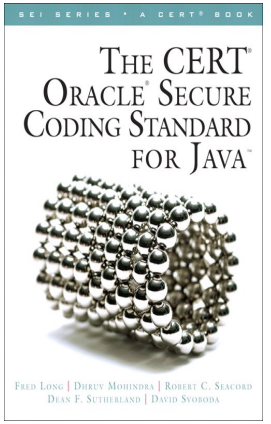
Apache Tomcat/7.0.2

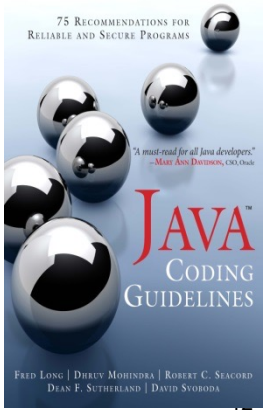
Ist das denn ein Problem?

- Schritt 1: Google Suche nach „apache tomcat 7.0.2 vulnerabilities“
- Schritt 2: Zweiten Treffer gewählt
- Schritt 3: Insgesamt 42 Schwachstellen gefunden!
- Darunter z.B.:
  - CVE-2013-4444 „Unrestricted file upload vulnerability in Apache Tomcat 7.x before 7.0.40, in certain situations involving outdated java.io.File code and a custom JMX configuration, allows remote attackers to execute arbitrary code by uploading and accessing a JSP file.“
  - CVE-2011-3190 „Certain AJP protocol connector implementations in Apache Tomcat 7.0.0 through 7.0.20, 6.0.0 through 6.0.33, 5.5.0 through 5.5.33, and possibly other versions allow remote attackers to spoof AJP requests, bypass authentication, and obtain sensitive information by causing the connector to interpret a request body as a new request.“
  - CVE-2011-1419 “Apache Tomcat 7.x before 7.0.11, when web.xml has no security constraints, does not follow ServletSecurity annotations, which allows remote attackers to bypass intended access restrictions via HTTP requests to a web application. NOTE: this vulnerability exists because of an incomplete fix for CVE-2011-1088.“

Sichere Programmierung

- Fast alle Sicherheitslücken sind auf schlechte, fahrlässige und unsichere Programmierung zurückzuführen!
- Deswegen müssen SW-Entwickler und –Architekten Literatur zur sicheren Programmierung lesen
- Beispiel Java
  - **Secure Coding Guidelines for Java SE**
    Updated for Java SE 11, Version: 7.2, Last updated: 27 September 2018
    http://www.oracle.com/technetwork/java/seccodeguide-139067.html
  - ***The CERT™ Oracle™ Secure Coding Standard for Java***
    Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda
    Rules available online at www.securecoding.cert.org
  - ***Java Coding Guidelines***
    Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda



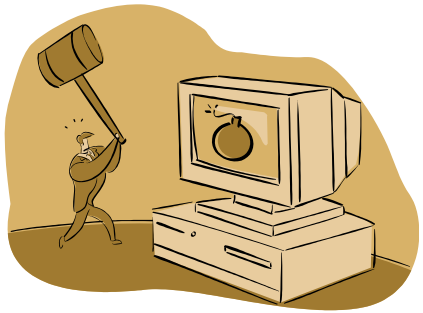


Regeln zur Software-Entwicklung

- Best Practices für ordentlichen Code
  - Vermeide doppelten Code
  - Isoliere Funktionalität
  - Begrenze die Länge von Codeblöcken
  - Begrenze die Funktionalität von Funktionen
  - Begrenze die Notwendigkeit für Kommentare
  - Begrenze den Gültigkeitsbereich von Variablen
  - Führe Unit-Tests durch
- Feinde des sicheren Codes
  - Ignoranz
  - Unordnung
  - Deadlines, Zeitdruck

Maßnahme für sichere Software: Test All Software!!!

- Permanente manuelle und automatische Tests sind notwendig um Fehler in der Software rechtzeitig zu erkennen und zu beseitigen
- Die Arten der Test-Verfahren ist vielfältig, hier sind ein paar Beispiele:
  - Unit-Test, Komponenten-Test, Integrations-Test, System-Test
  - Akzeptanztest, Test Driven Development
  - Usability-Test
  - Regressionstests
  - Lasttest, Robustness Testing
  - Fuzz Testing (Fault Injection), Penetrationstest



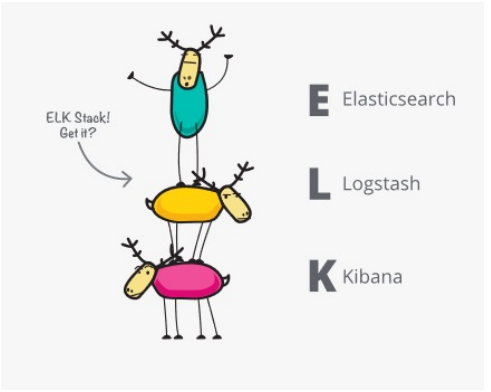
- Zusätzlich muss man regelmäßig automatische Analysen über die Software mit Tools machen
  - Static Application Security Testing (**SAST**)
  - Dynamic Application Security Testing (**DAST**)

Ein Restrisiko bleibt immer, deshalb ist Logging erforderlich

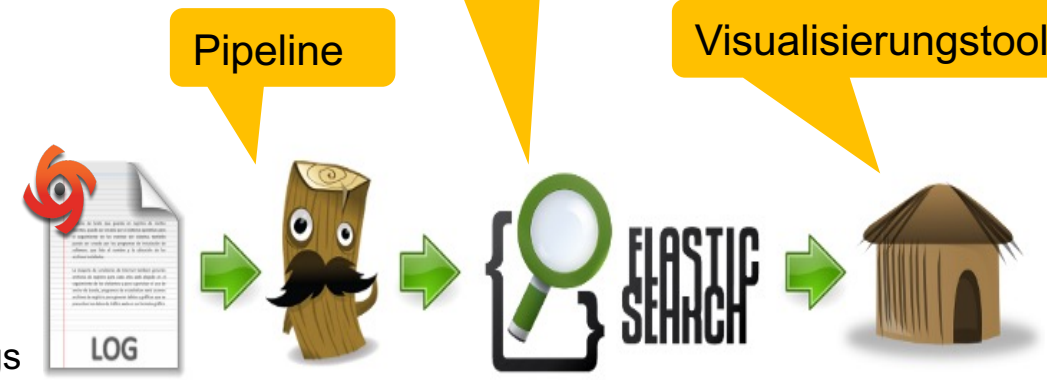
- Ein Restrisiko bleibt immer: Aus diesem Grund müssen Angriffe nicht nur abgewehrt sondern auch erkannt werden.
- Wichtigste Informationsquelle zur Erkennung von Angriffen und Eindringlingen sind Logs.

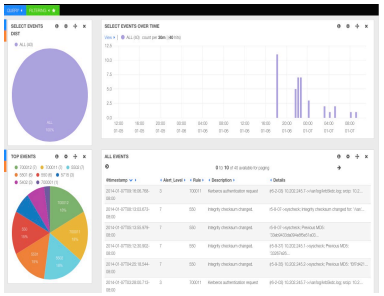
  Gutes Logging ist die Basis für Forensik
- Zentrale Fragen:
  - Was wird geloggt? (https://www.owasp.org/index.php/Logging_Cheat_Sheet)
  - Was wird nicht geloggt?
  - Wie werden die Logs eingesammelt? (z.B. logstash, …)
  - Wie werden die Logs analysiert und Visualisiert? (z.B. Kibana, …)
- Es gibt vorgefertigte Tool-Ketten zur Log-Analyse (ELK Stack)
  z.B. von OSSEC (https://www.ossec.net), elastic (https://www.elastic.co/de)

Der ELK Stack



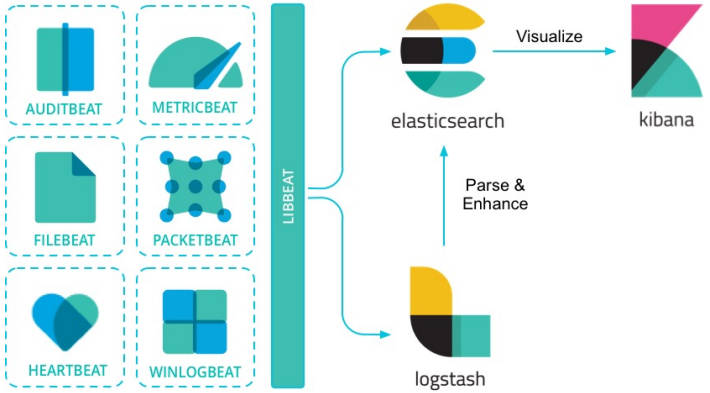
- Server-Logs
- Anwendungs-Logs
- Schnittstellen-Logs
- Betriebssystem-Logs





- Abnorme Trends
- Angriffsmuster
- Alerts

Daten Shipper



Elastic stack

Quelle: https://www.elastic.co/de/what-is/elk-stack

Regeln zu Auditing, Logging

- **Auditing**
  - Aufzeichnung wichtiger **Benutzer**aktivitäten (Nachweisbarkeit)
- **Logging**
  - Protokolliere alle **sicherheitsrelevanten** Ereignisse
- Identifiziere bösartiges Verhalten
- Kenne das normale Verhalten der Applikation (good traffic)
- Auditiere und Logge Aktivitäten auf allen Applikationsschichten
- Schränke den Zugriff auf Log-Dateien ein
- Keine privaten Daten (wie z.B. Passwörter) in Log-Dateien
- Sichere und analysiere die Log-Dateien regelmäßig

Sichere Administration und Betrieb

- Neben sicherere Programmierung ist auch ein sicherer Betrieb erforderlich.
- Ziel: Reduktion der Angriffsvektoren
- Aufbau einer sicheren Infrastruktur (FW, WAF, SSL-Gateway, Access Gateway, Security Zones, ...)
- Abschottung der Produktivsysteme (strikte Trennung von Test- und Entwickung-Systemen, insbesondere Credentials)
- Sichere die Remote-Zugänge für Administration besonders stark
- Etabliere ein Incidence Resonse und Notfall Management
- Change Management
  - Viele Sicherheitslücken kommen nachträglich in die Systeme

Sichere Administration und Betrieb

- **System Hardening**: Konfiguriere alle eingesetzten Systeme richtig
  - Patches einspielen
  - Unsichere Default-Einstellungen ändern
  - Zugriffsrechte anpassen
  - Deaktivierung unnötiger Dienste
  - Vulnerability scans
- **Decomission**
  - Entferne Authentifizierungs- und Autorisierungseinstellungen (accounts, key, certificates, …)
  - Reset Configuration settings (FW settings, open ports, DNS entries, …)
  - Lösche oder sichere die Daten

Sicherheit in Cloud Plattformen

- In Cloud Plattformen gibt es die Möglichkeit von **Continuous Integration CI** und **Continuous Deployment CD**
- Die Agilität steigt, aber auch die Sicherheitsrisiken
- Die Bereiche Entwicklung und Betrieb wachsen in modernen Cloud Plattformen basierend auf Micro Services immer mehr zusammen
- Entwickler müssen sich auch mit Administration und Betrieb auseinander setzen -> es gibt eine neue Rolle im Software Engineering **DevOps**
- In Cloud Plattformen gilt: Automatisiere so viel wie möglich
  - Das sorgt für Nachvollziehbarkeit was in der Plattform abläuft
  - Es reduziert Fehler und die menschliche Interaktion

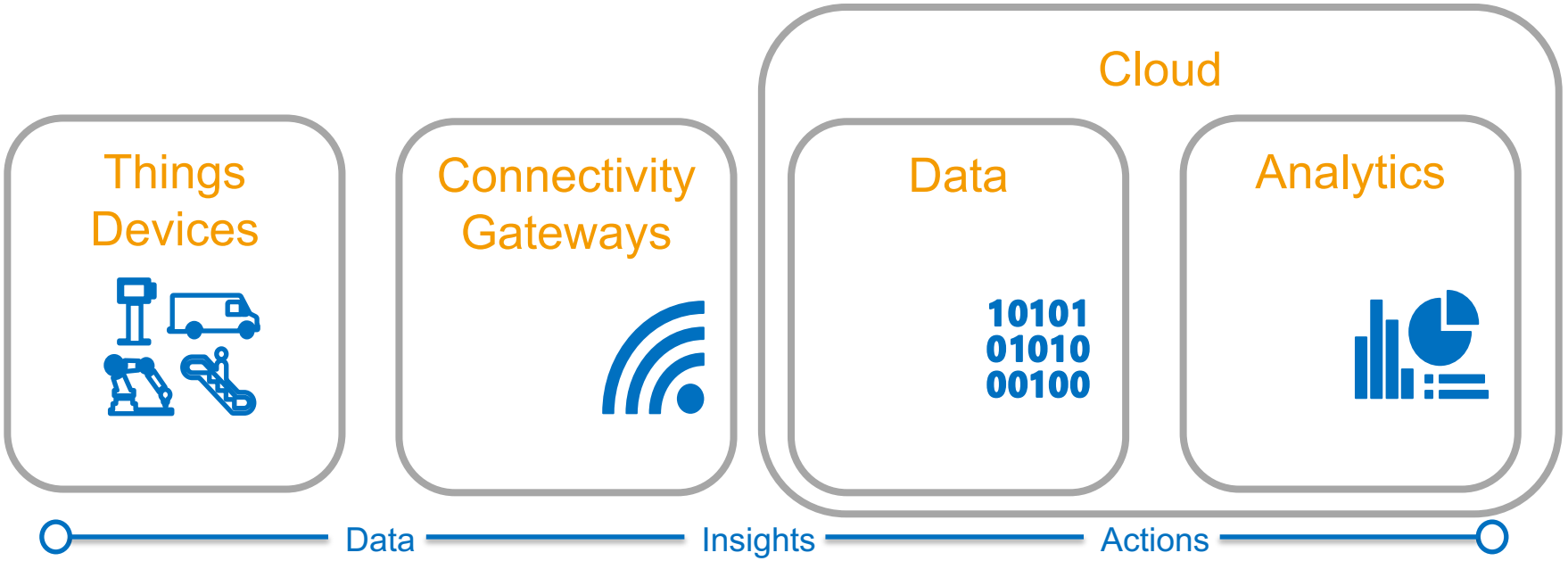
DevOps/Cloud Security Issues

- Absicherung der CI/CD Pipeline
  - Ziel: Deployment von vertrauenswürdigen Quellen
  - Risiko: Ausfälle wegen ungetesteten oder unsicheren Code
  - Maßnahmen: Version control in GitHub, 2FA, 4-Eye-Principle, automatic tests, security checks, artifact repository, image registry, rollback options,
- Image und Container Security
  - Risiko: Unautorisierte Änderungen auf der Prod-Umgebung
  - Maßnahmen: verwende sicher Base Images/Containers, Vulnerability checks für verwendete libraries, external services, APIs, Frameworks, Konfigurationen
- Networksecurity
  - Absicherung gegen andere Applikationen, Mandanten, Angreifer auf der Plattform
  - Isolation
  - Ingress and Egress control

DevOps/Cloud Security Issues

- Access and Account Management
  - Kontrolle der Rechte und Rollen auf verschiedenen Ebenen
  - Technical Accounts (Administration der Cloud Plattform und Infrastruktur, GitOps Cockpit auf Basis Cluster Security Policies)
  - Fachliche Accounts (Administration der Anwendung, User Management, Data Services, Reporting, Billing, Hotline)
  - DevOps/Wartungs Accounts (Error Analysis and Recovery, Logging, Monitoring, Alerting)
- Secret Management
  - Ziel: Passwörter, API Keys, Passwörter, Zugriffstoken müssen geschützt werden
  - Maßnahmen: Sealed Secrets, Key Vault, Secret Manager
- Cluster Backup and Recovery
  - Verhinderung von Datenverlust
  - Schneller Failover

IoT-Sicherheit (Internet of Things)



- IoT bedeutet: viele Geräte, viel Kommunikation, viele Daten
- Beispiele für IoT Anwendungen
  - Smart Home, Smart Mobility, Smart Factory, Smart Health

IoT bringt neue Herausforderungen für die IT-Sicherheit

- Absicherung von Geräten die keinen physischen Zugriffsschutz haben
- Aktualisierung von (alten) Geräten für einen langfristigen Betrieb
- Große Datenmengen die Bewegungs- und Verhaltenskontrolle ermöglichen
- Autonome Systeme treffen eigenständige Entscheidungen
- Massive Kommunikation und Weitergabe von Daten
- Wenig Standards mit Sicherheitsfeatures
- Eine große Menge an z.T. billigen Konsumerprodukten

Quelle: Wendzel S., IT –Sicherheit für TCP/IP- und IoT-Netzwerke, Springer-Vieweg, 2018, eBook in Bibliothek

IoT hat besondere Risiken

- Bedrohungen
  - Hijacked Devices (IoT Botnets)
  - Informationsabfluss
  - Störung von Diensten, Funktionalität
- Schwachstellen
  - Exposed Device in WWW (Shodan tool zeigt exposed devices https://www.shodan.io/)
  - Insecure Webinterfaces (bad session management, default credentials)
  - Insecure Frameworks and protocolls
  - Generic security vulnerabilities (no access control, no logging, storage not secured, hardcoded passwords
  - Outdated devices
- IoT erfordert mehr Skills als reines Softwareengineering: zusätzlich ist Systemengineering erforderlich

Sicherheitsmaßnahmen für IoT

- Für IoT gelten alle Regeln zur sicheren Applikationsentwicklungen.
- Ein besonderer Fokus ist auf folgende Maßnahmen zu setzen
  - Sämtliche Kommunikation verschlüsseln
  - Secure Key Management
  - (Mutual) Authentication Mechanism
  - Keine Ausführung von unautorisierten Code
    - Sichere Updates
    - Code Signing Mechanismus
  - Devices als Clients implementieren (keine Services hosten, nicht als Server agieren)
  - Least privilege
  - Intrusion detection

OWASP für IoT

I1 Weak Guessable, or Hardcoded Passwords

I2 Insecure Network Services

I3 Insecure Ecosystem Interfaces

I4 Lack of Secure Update Mechanism

I5 Use of Insecure or Outdated Components

I6 Insufficient Privacy Protection

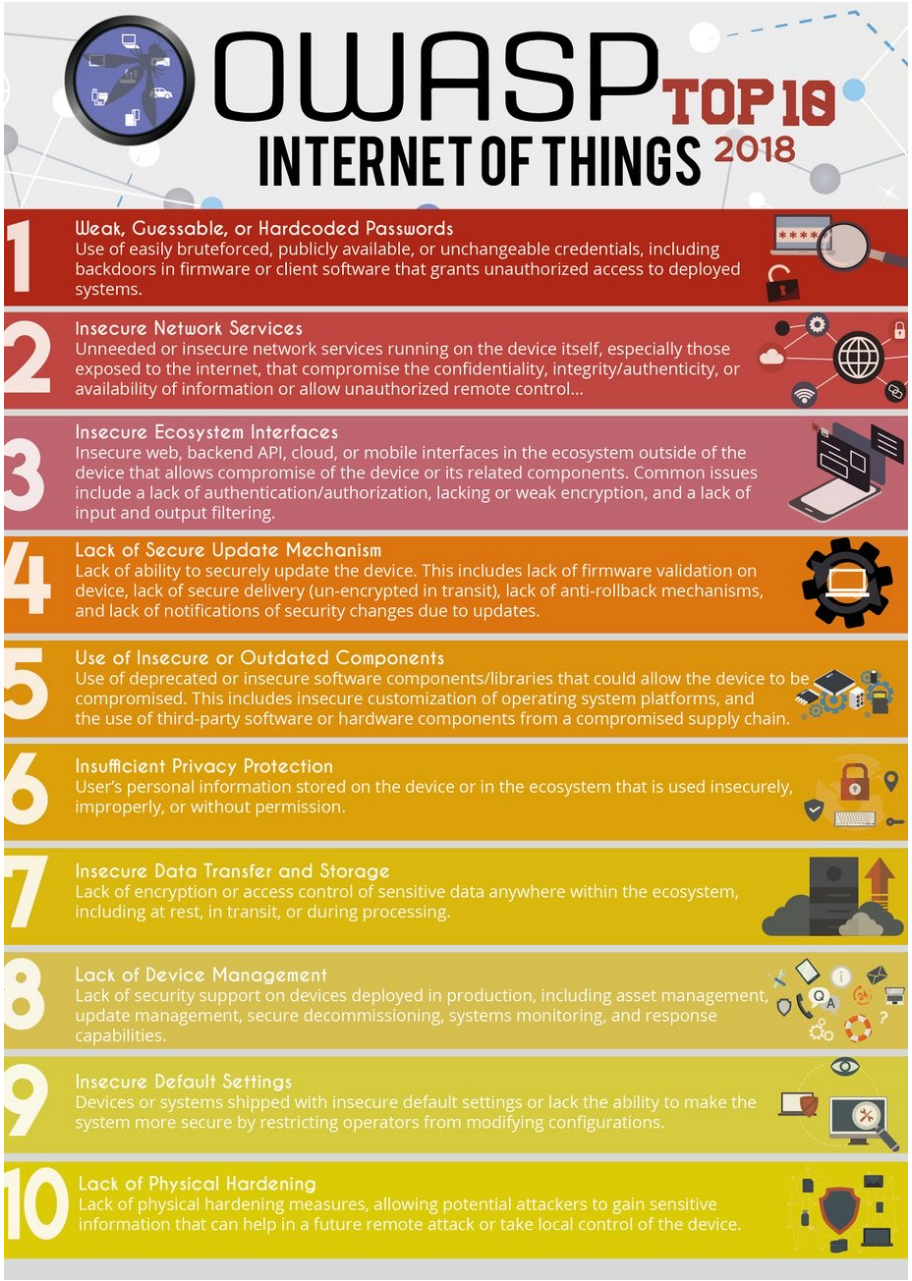
I7 Insecure Data Transfer and Storage

I8 Lack of Device Management

I9 Insecure Default Settings

I10 Lack of Physical Hardening

https://owasp.org/www-project-internet-of-things/



Zusammenfassung Applikationssicherheit

- Jeder SW-Engineer muss die wichtigsten Bedrohungen kennen und seine Applikationen dagegen schützen.
- Security muss von Anfang an in dem SW-Engineering Prozess berücksichtigt werden.
- Alle Eingaben an eine Applikation müssen validiert, kanonisiert, gefiltert, gereinigt und escaped werden.
- Zusätzlich müssen Maßnahmen wie sichere Programmierung, Logging, sichere Fehlerbehandlung, Sicherheitstests und sicherer Betrieb umgesetzt werden
- In bestimmten Anwendungsdomänen (z.B. Cloud, IoT) müssen zusätzliche Maßnahmen beachtet werden