IT-Sicherheit

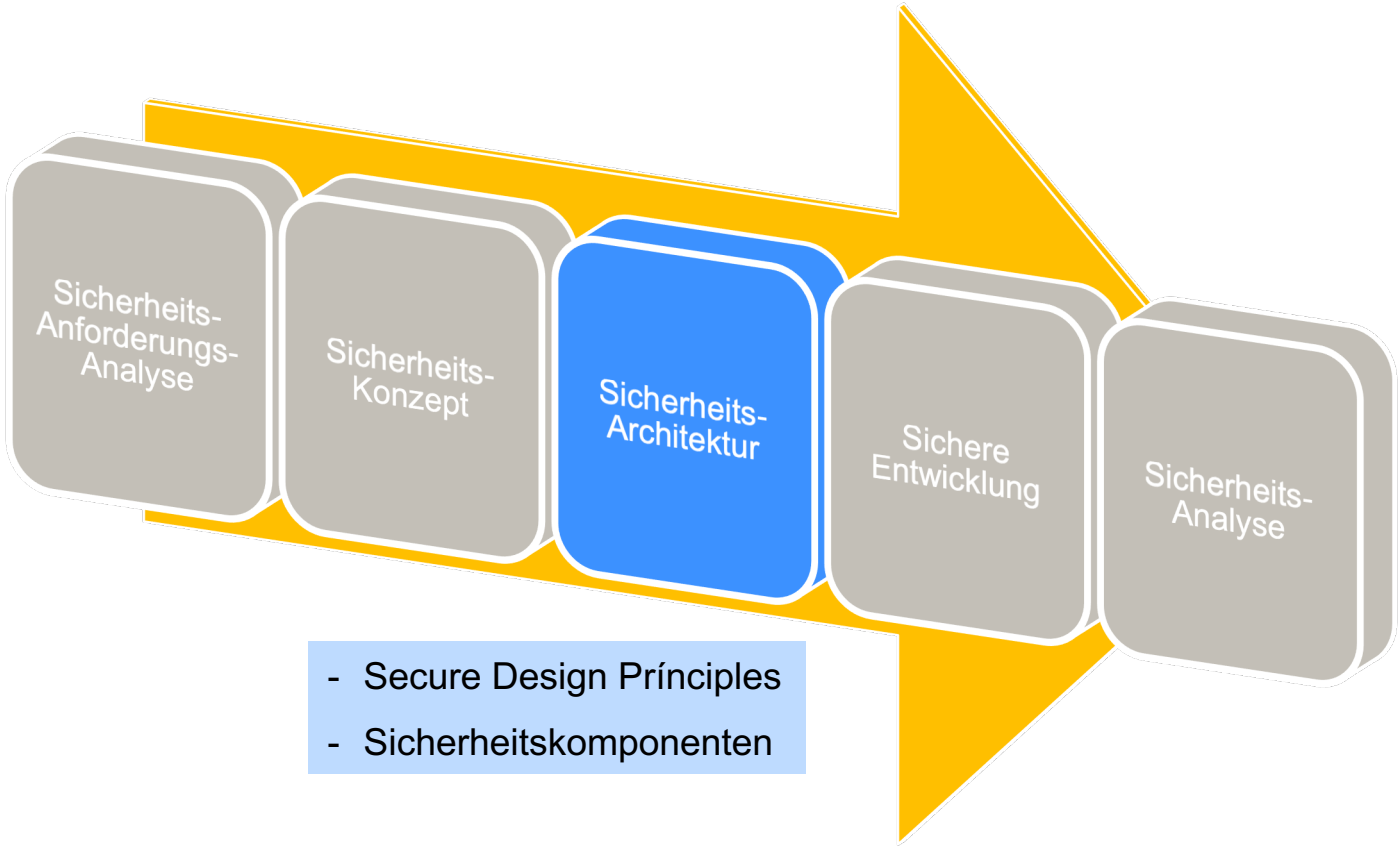
Kapitel 6: Secure Software Engineering

Teil 2

- Sicherheitsarchitektur und- Design
- Tools zur Sicherheitsanalyse

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

Sicherheitsarchitektur und -Design



- Secure Design Prínciples
- Sicherheitskomponenten

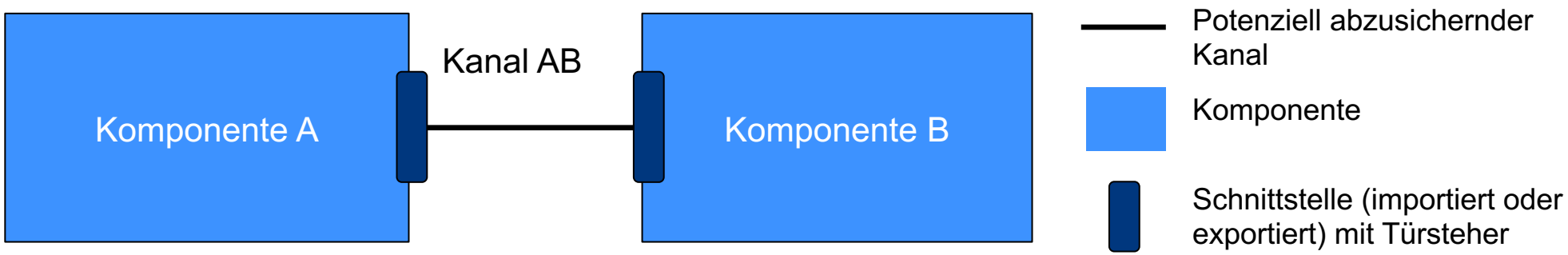
Secure Design Principles

- In der Design Phase gibt es eine Menge allgemeiner Prinzipien die zu sicheren Software Systemen führen
- Viele dieser Prinzipien sollten auch in den anderen Entwicklungsphasen berücksichtigt werden
- Es gibt viele Quellen und Prinzipien, z.B.
  - OWASP https://github.com/OWASP/DevGuide/blob/master/02-Design/01-Principles%20of%20Security%20Engineering.md
  - https://www.ncsc.gov.uk/collection/cyber-security-design-principles
  - http://www.opensecurityarchitecture.org/cms/foundations/design-principles
- Es gibt sogar eine generische Schwachstelle zu diesem Thema
  - CWE-657: Violation of Secure Design Principles https://cwe.mitre.org/data/definitions/657.html

Secure Design Principles

1. Don’t trust third parties
2. Defense in depth
3. Keep it simple
4. Secure Defaults
5. Open design
6. Fail safe
7. Least privilege
8. Minimize attack surface
9. Know your enemies
10. Psychological acceptability

Sicherheitskomponenten, Kanälen und Schnittstellen



- Jedes System/Systemverbund besteht aus Komponenten, die über Kanäle und Schnittstellen verbunden sind
- Die Sicherheitsarchitektur besteht aus Sicherheitskomponenten mit potenziell abzusichernden Kanälen und Schnittstellen.

Jede Komponente hat eine definierte Sicherheit

- Sicherheitskomponenten
  - Sicherheitskomponenten können sein: Eine Hardware-Einheit, ein Rechenzentrum, eine VM, ein App-Server, eine DB, …
  - Jede Sicherheitskomponente wird von jemand verantwortet (Privatmann, RZ-Betreiber, ..) und benutzt.
    Dieser Jemand ist vertrauenswürdig oder auch nicht.
  - Jede Sicherheitskomponente hat eine definierte Sicherheit (von sehr sicher bis unsicher)

- Schnittstellen/Türsteher
  - Wie gründlich müssen die *Türsteher* an ihren Ein- und Ausgängen sein? Von der Festung bis zum Jedermanns-Recht.

- Kanäle
  - Jeder Kanal hat eine definierte Sicherheit (von sehr sicher bis unsicher)
  - Wie robust ist der Kanal und das dabei verwendete Protokoll gegenüber den typischen Angriffen?

An den Schnittstellen sind Türsteher

- Der Türsteher stellt sicher, dass nur saubere Nachrichten in die Anwendung gelangen.
- Seine typischen Aufgaben dabei sind:

Sicherheitsziel	Aufgabe / Anforderung	Mögliche Lösungen
Vertraulichkeit	Nur sicheren Kanal akzeptieren	• HTTPS mit Perfect Forward Secrecy • Verschlüsselung auf Anwendungsebene
-- " --	Prüfen, ob der Absender die Nachricht auch senden darf	• URLs gegen ACL prüfen (.htaccess) • Anwendungsebene: Inhalt der Nachricht
Authentizität	Echtheit des Absenders prüfen	Authentifizierung (z.B. über Client-Zertifikat oder Basic Auth)
-- " --	Aktualität (Freshness) der Nachricht prüfen	Nonces (Einmal-IDs) im Response vergeben und im Request prüfen
Integrität	Inhalt der Nachricht prüfen und bereinigen	Validierung: So exakt wie möglich. Anwendungsspezifisch. Syntaktisch und semantisch. (Kryptographische) Prüfsummen
Verfügbarkeit	DoS-Angriffe erkennen und abwehren	Rate Limiting und Tarpits.
Nicht-Abstreitbarkeit	Inhalt einer Nachricht versiegelt ablegen	Signatur und Signatur-Prüfung

Türsteher können technische Bausteine nutzten

- **Security Frameworks**
  wie z.B. OWASP ESAPI
  bieten eine umfassende Sammlung an vorgefertigten Sicherheitsfunktionen zur Integration in Web-Anwendungen, wie:
  - Authentifizierung und Single-Sign-On
  - Autorisierung
  - Kryptographie
  - Session Management
  - Abwehr von Angriffen

OWASP Top Ten	OWASP ESAPI
A1. Cross Site Scripting (XSS)	Validator, Encoder
A2. Injection Flaws	Encoder
A3. Malicious File Execution	HTTPUtilities (upload)
A4. Insecure Direct Object Reference	AccessReferenceMap
A5. Cross Site Request Forgery (CSRF)	User (csrftoken)
A6. Leakage and Improper Error Handling	EnterpriseSecurityException, HTTPUtils
A7. Broken Authentication and Sessions	Authenticator, User, HTTPUtils
A8. Insecure Cryptographic Storage	Encryptor
A9. Insecure Communications	HTTPUtilities (secure cookie)
A10. Failure to Restrict URL Access	AccessController

- **Web Application Firewalls**
  wie z.B. mod_security schützen den HTTP-Kanal bereits bevor ein Request beim Server ankommt mit:
  - Rate Limiting und Tarpits
  - Autorisierung und Authentifizierung
  - Abwehr von klassischen Angriffen

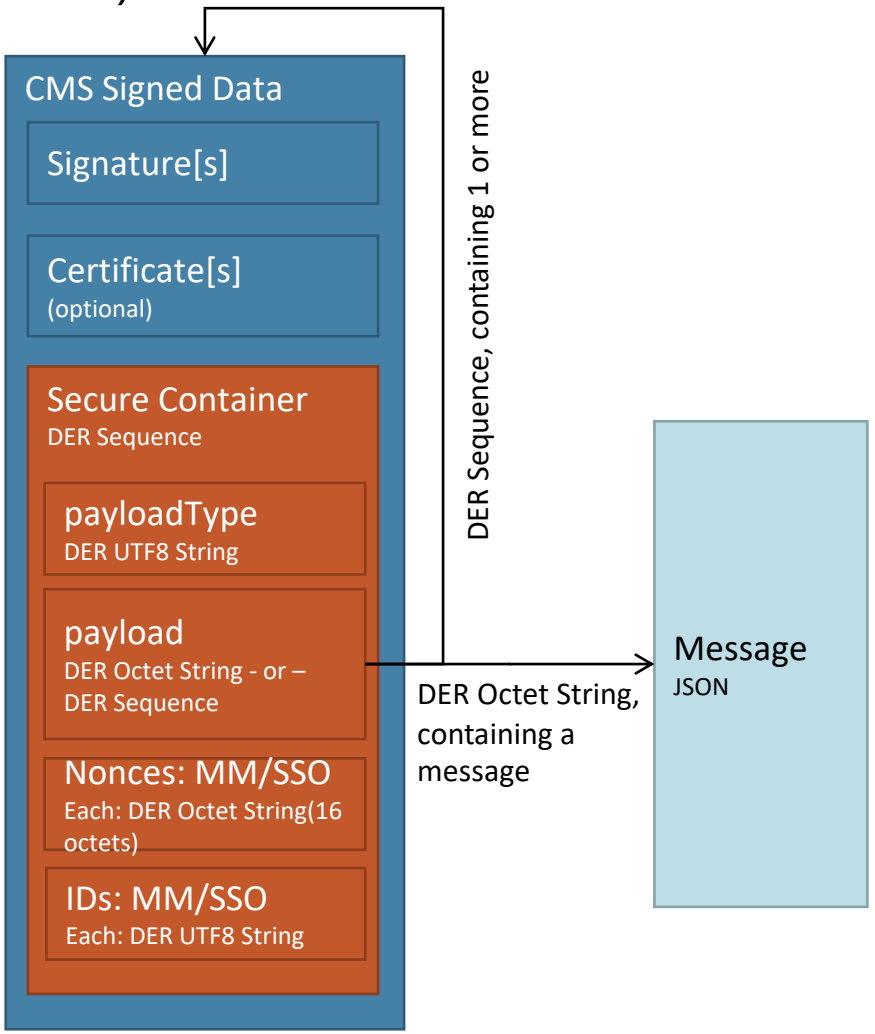
Die restlichen Prüfungen sind von fachlicher Natur

- Der Türsteher weiß, welche Nachricht
  - fachlich erwartet wird, und lässt nur die richtigen durch
  - von welchem Absender erwartet wird, und an welchen Adressaten sie gehen darf
  - nach welchen Regeln validiert wird
  - nach welchen Regeln gesichert sein muss
  - Er verzahnt die Ebenen der Kommunikations-Kanäle (*Deep Inspection*).

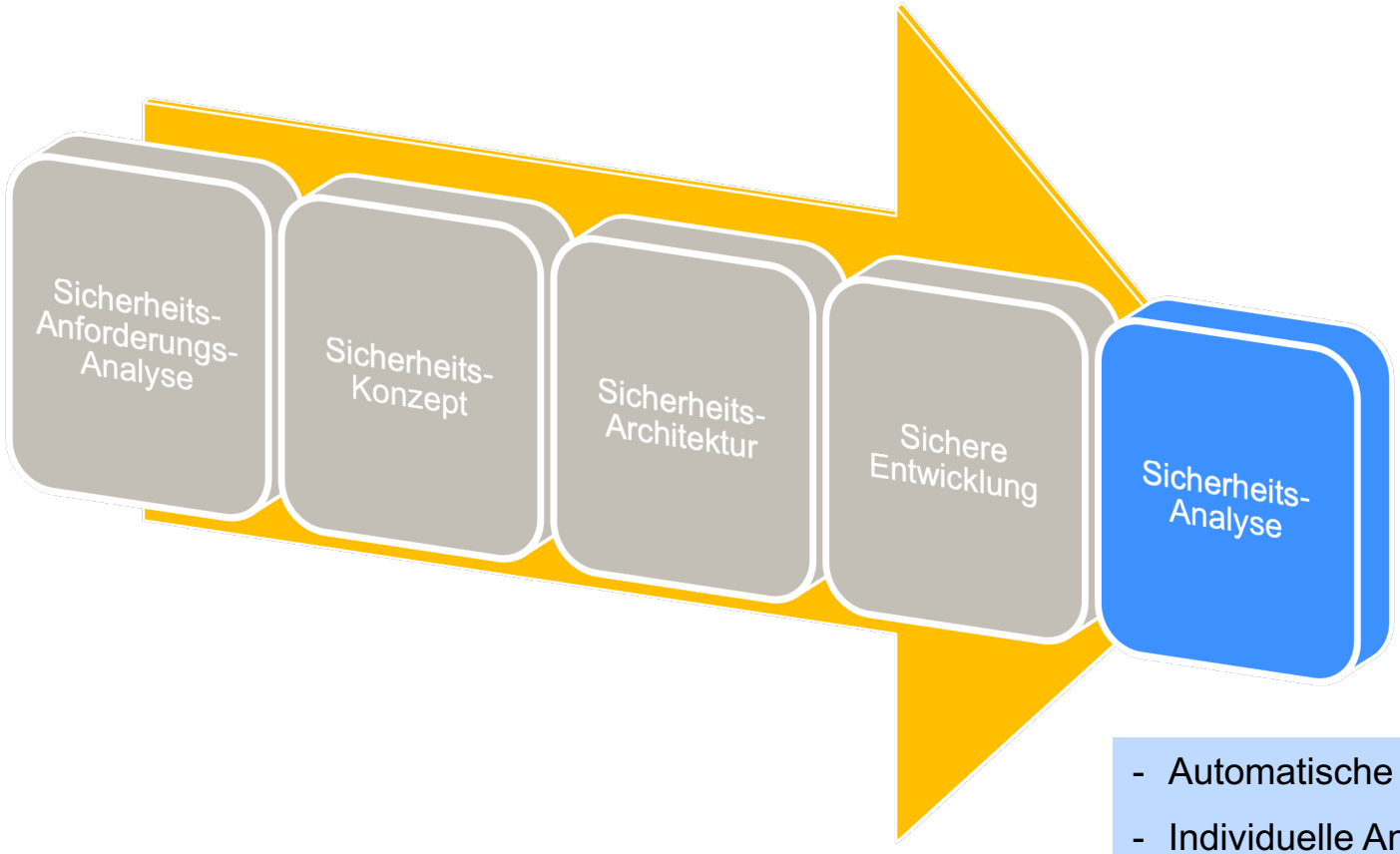
- Den Payload-Type sollte man explizit angegeben
  - Nicht raten, sondern wissen und dagegen prüfen
  - Erst PayloadType prüfen: Passt das zur Erwartung?
  - Dann Payload parsen: Ist das wirklich der angegebene Typ?

Beispiel für ein sicheres Nachrichtenformat

- CMS Cryptographic Message Syntax (PKCS#7)
  - Nachrichten signieren
  - Nachrichten verschlüsseln
  - Der Standard, kann mit den bekannten Bibliotheken geprüft werden,
    z.B. OpenSSL



Sicherheitsanalyse



- Automatische Analysen
- Individuelle Analysen

Sicherheitsanalyse – Automatisierte Analysen

- SAST Static Application Security Testing
  - Automatisierte statische Analyse Tools
  - Liefern zur Compile-Zeit Warnungen wo Fehler auftreten können
  - Bsp.: SonarQube, FindBugs, Fortify

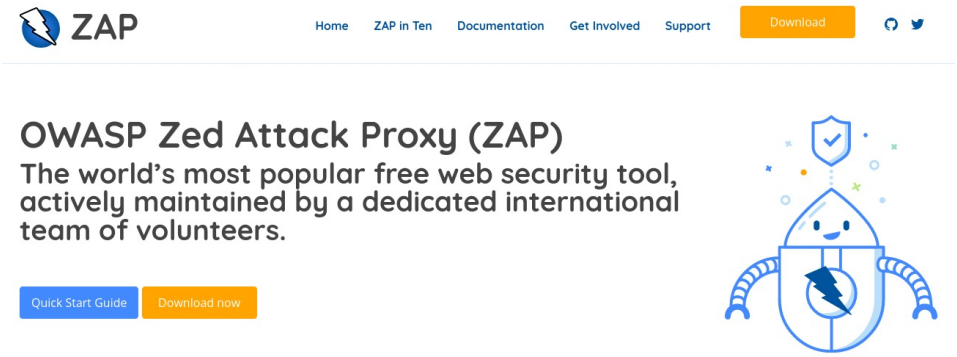
- DAST Dynamic Application Security Testing
  - Web Security Scanner, Vulnerability Scanner: Computerprogramme die Zielsystem auf bekannte Sicherheitslücken untersuchen, z.B. OWASP ZAP, Nessus, Qualys)
  - **Passive Scans**
    - durchsuchen die Daten zwischen Browser und Webserver nach Muster
    - ändern weder den Request noch die Response,
    - findet im Hintergrund statt
  - **Aktive Scans**
    - suchen Verwundbarkeiten durch Anwendung von bekannten Attacken
    - **darf nicht auf fremde Web Anwendungen ohne Erlaubnis angewandt werden**

Individuelle Analyse Methoden

- **Penetrationstest**
  - Überprüfung aller Systembestandteile mit Mittel und Methoden die ein Angreifer (Hacker) anwenden würde
  - Blackbox-, Whiteboxtest
  - Muss geplant werden (Scoping, Vorbereitung, Voranalyse, Analyse, Debriefing, Retest)
  - Muss von Profis gemacht werden
- **Fuzz Testing (Fault Injection)**
  - Erzeugt (halb) automatisiert ungültige, unerwartete oder zufällige Eingabedaten
  - Beobachtet Ausnahmen, Abstürze oder fehlende Fehlerbehandlung (Monitoring)
- **Security Code Review**
  - Gegenseitige Reviews sind zusätzlich zu SAST notwendig

Tools zur Sicherheitsanalyse

- Attack Proxies – Tools zur dynamischen Analyse
  - Nessus, Qualys, Burp Suite
  - OWASP ZAP



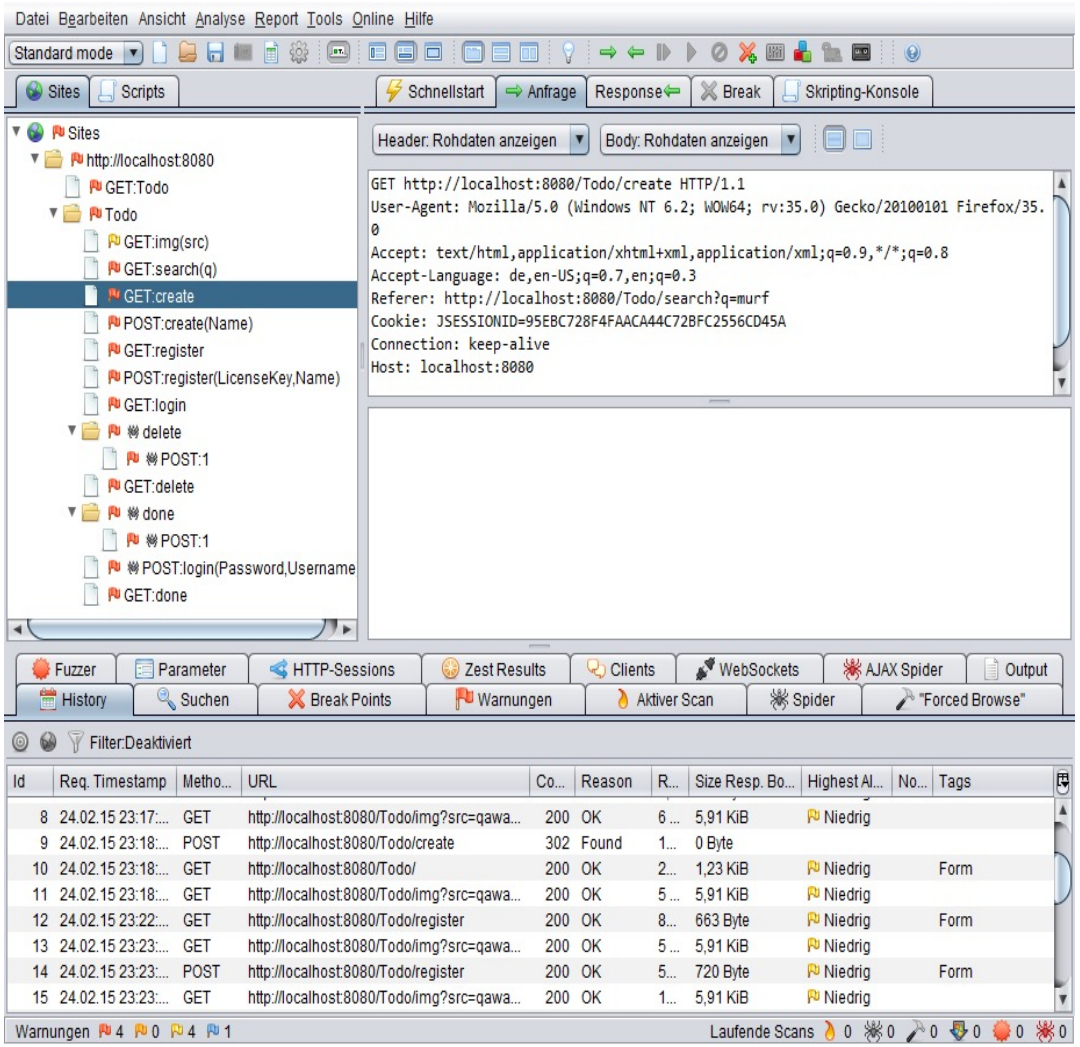
https://www.zaproxy.org/

- OWASP Dependency Check
  - Software Composition Analysis (SCA) tool that attempts to detect publicly disclosed vulnerabilities contained within a project's dependencies

OWASP ZAP - ZAP Attack Proxy

https://owasp.org/www-project-zap/

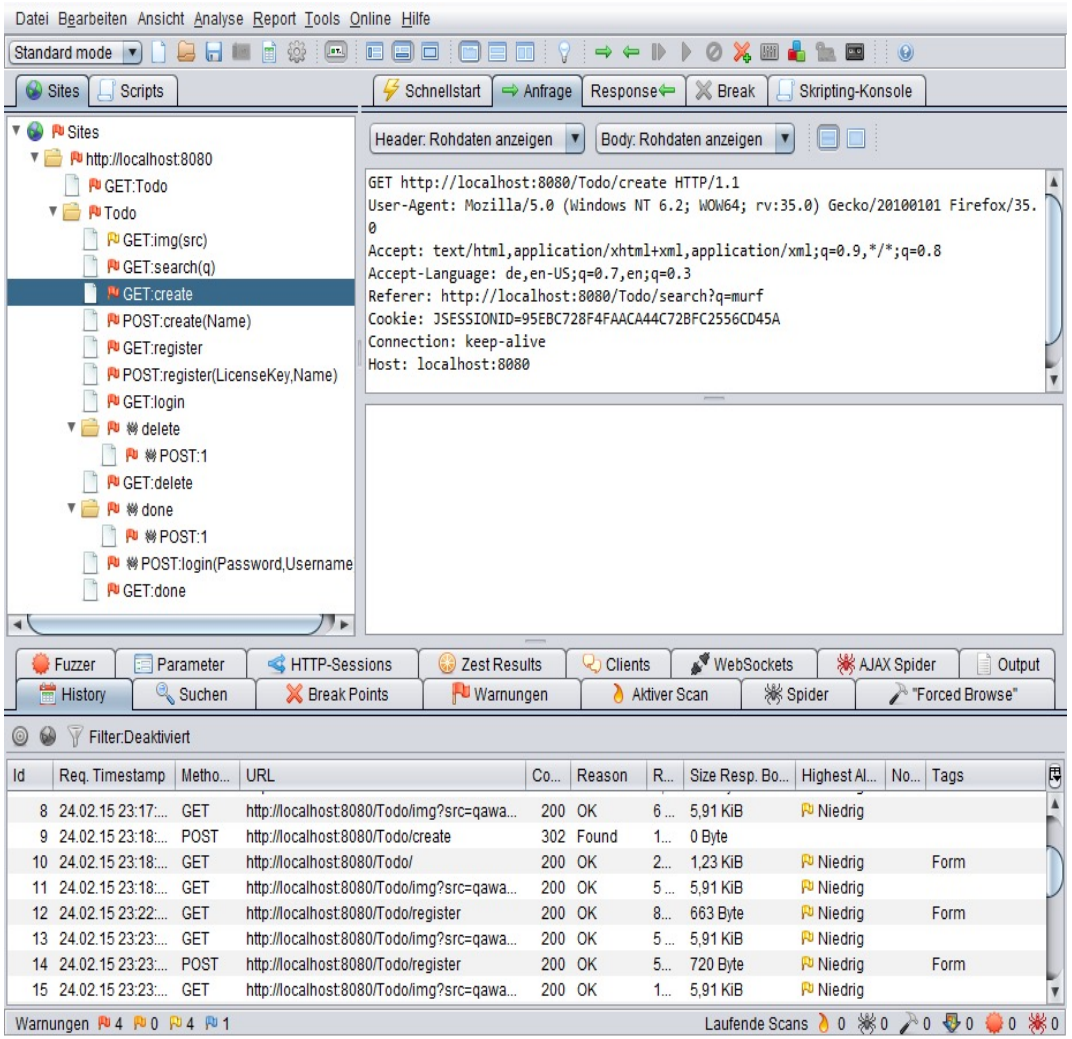
- Open Source Java Tool mit sehr aktiver Community
- Attack Proxies sind ein Standardwerkzeug für Pentester
- ZAP ist explizit nicht nur für PenTester sondern auch für Entwickler gedacht
- Erweiterbar durch Plugins



OWASP ZAP - ZAP Attack Proxy

Features:

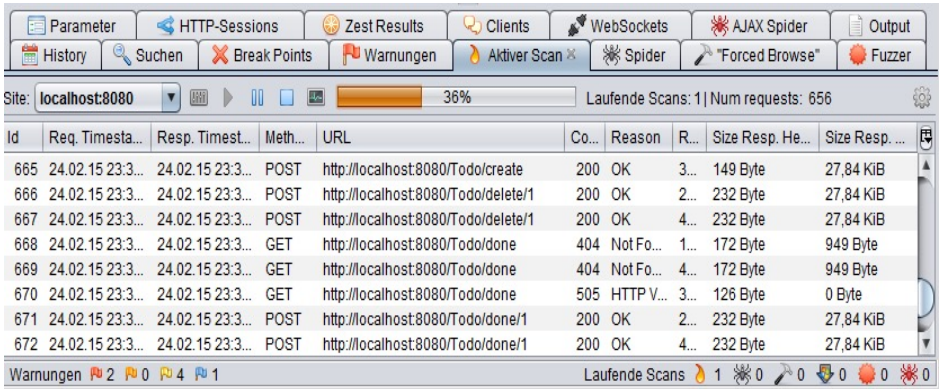
- Intercepting Proxy
  - Schaltet sich zwischen Browser und zeichnet Requests mit auf
  - On-the-Fly anpassen von Request/Response durch Breakpoints
- Spider
  - Automatisches entdecken von neuen URLs durch parsen der Responses

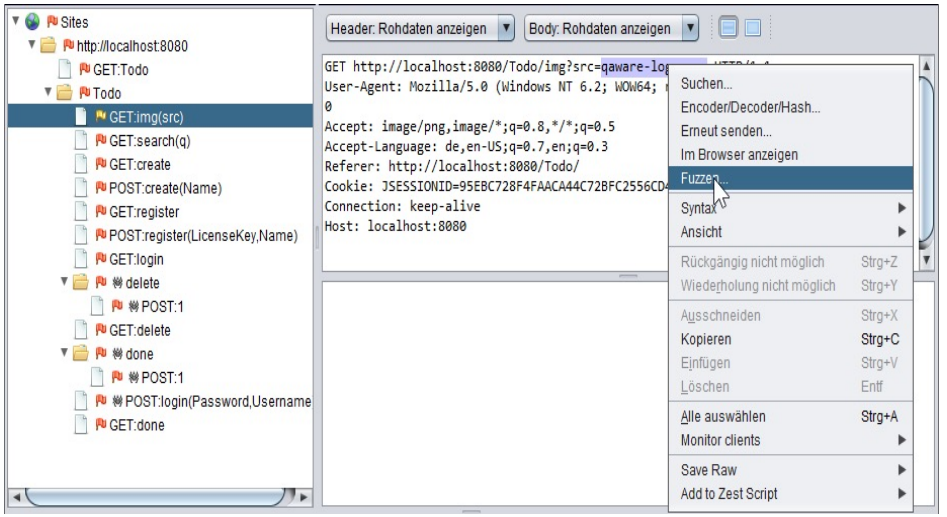


OWASP ZAP - ZAP Attack Proxy

Features II:

- Aktiv und passiv Scanner
  - Passiv-Scan: Scannt alle Responses auf Schwachstellen
  - Aktiv-Scan: Vollautomatischer Scan der Anwendung
- Fuzzer
  - Ersetzt Strings in Requests durch bekannte Angriffsmuster





OWASP ZAP – Unbedingt beachten!

- Scans nur auf Systemen ausführen, wenn ich die **explizite Erlaubnis** des Besitzers habe
- Aktive Scans **nur auf Testsystemen** ausführen

Hackerparagraph im Strafgesetzbuch

§ 202c Vorbereiten des Ausspähens und Abfangens von Daten

https://dejure.org/gesetze/StGB/202c.html

Wer eine Straftat nach § 202a oder § 202b vorbereitet, indem er

1. Passwörter oder sonstige Sicherungscodes, die den Zugang zu Daten (§ 202a Abs. 2) ermöglichen, oder
2. Computerprogramme, deren Zweck die Begehung einer solchen Tat ist,

herstellt, sich oder einem anderen verschafft, verkauft, einem anderen überlässt, verbreitet oder sonst zugänglich macht, wird mit Freiheitsstrafe bis zu einem Jahr oder mit Geldstrafe bestraft.

- **§ 202a** Ausspähen von Daten
- **§ 202b** (Abfangen von Daten)

- Bloße Vorbereitung und Besitz von Hackertools ist bereits strafbar!

OWASP TOP 9
Using Components with known Vulnerabilities

- Problem: Sicherheit wird meist durch die richtige Benutzung des Frameworks erreicht
  - Prepared Statements in JDBC / Hibernate
  - OAuth mit SpringBoot
  - Access-Token-Vergabe durch Web-Container
  - etc

- Was, wenn eines dieser Frameworks eine Sicherheitslücke hat?

CWE / CVE – Die Datenbank für bekannte Schwachstellen

- Betrieben von Mitre.org
  - Non-Profit-Organization aus den USA
  - Ziel: Bündelung von Spezialwissen aus Informationstechnologe, Betriebskonzepte und Unternehmenssteuerung
- **CWE** - Common Weakness Enumeration
  - Sammlung generischer Schwachstellen
- **CVE** – Common Vulnerability Enumertation
  - Sammlung produktspezifischer Verwundbarkeiten
- **CVSS** – Common Vulnerability Scoring System
  - Schema zur Bewertung von Verwundbarkeiten nach Kritikalität

CWE - Common Weakness Enumeration

- Sammlung generischer Schwachstellen
  - Produktunabhängig
- http://cwe.mitre.org
- Wichtige Informationen:
  - Beispiele
  - Relationships – Wechselbeziehungen mit anderen Schwachstellen
  - References – Weitere Literatur zu einer Schwachstelle
  - Detection Mechanisms – Maßnahmen, wie die Schwachstelle aufgedeckt werden kann
  - Mitigations – Maßnahmen, wie die Schwachstelle verhindert werden kann

https://cwe.mitre.org/data/definitions/699.html

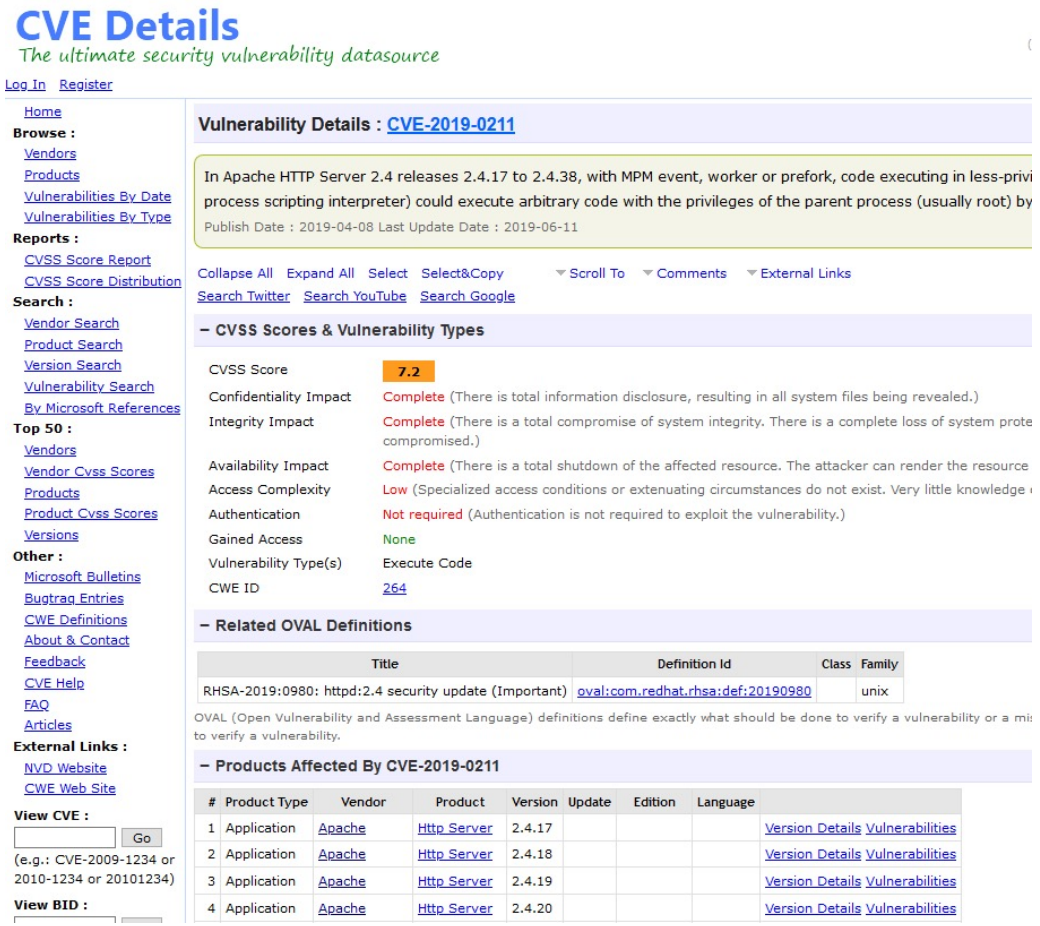
The CWE – Top 25 2019

Rank	ID	Name	Score
[1]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	75.56
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.69
[3]	CWE-20	Improper Input Validation	43.61
[4]	CWE-200	Information Exposure	32.12
[5]	CWE-125	Out-of-bounds Read	26.53
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	24.54
[7]	CWE-416	Use After Free	17.94
[8]	CWE-190	Integer Overflow or Wraparound	17.35
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	15.54
[10]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.10
[11]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	11.47
[12]	CWE-787	Out-of-bounds Write	11.08
[13]	CWE-287	Improper Authentication	10.78
[14]	CWE-476	NULL Pointer Dereference	9.74
[15]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.33
[16]	CWE-434	Unrestricted Upload of File with Dangerous Type	5.50
[17]	CWE-611	Improper Restriction of XML External Entity Reference	5.48
[18]	CWE-94	Improper Control of Generation of Code ('Code Injection')	5.36
[19]	CWE-798	Use of Hard-coded Credentials	5.12
[20]	CWE-400	Uncontrolled Resource Consumption	5.04
[21]	CWE-772	Missing Release of Resource after Effective Lifetime	5.04
[22]	CWE-426	Untrusted Search Path	4.40
[23]	CWE-502	Deserialization of Untrusted Data	4.30
[24]	CWE-269	Improper Privilege Management	4.23
[25]	CWE-295	Improper Certificate Validation	4.06

https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html

CVE – Common Vulnerability Enumeration https://cve.mitre.org/

- Sammlung Produktspezifischer Verwundbarkeiten von
  - Betriebssystemen
  - Programmen
  - Sprachen
  - Frameworks
  - etc
- Namensschema der Einträge:
  CVE-<Jahr>-<LaufendeNummer>
- Achtung: Nicht alle Verwundbarkeiten werden hier aufgelistet, aber insgesamt vollständigste Sammlung



https://www.cvedetails.com/cve/CVE-2019-0211/

CVSS – Common Vulnerability Scoring System

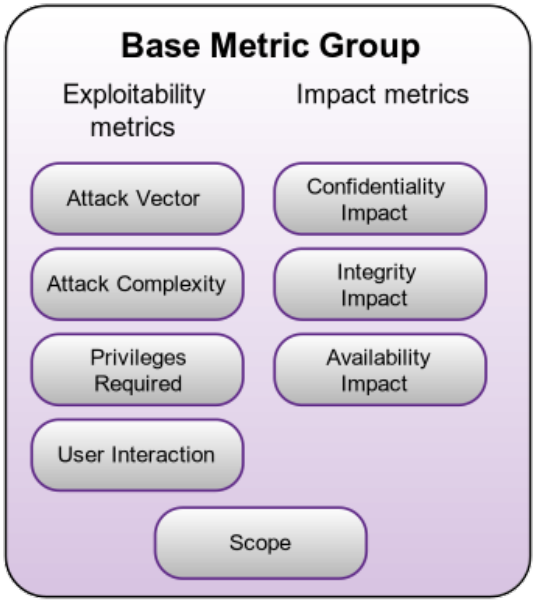
- Score von 0.0 (unkritisch) bis 10.0 (extrem kritisch)
- Achtung: CVSS ist ein erster Anhaltspunkt, kann aber nicht eine individuelle Einstufung im Projektkontext ersetzen
- In den Score fließen die verschiedene Parameter mit ein

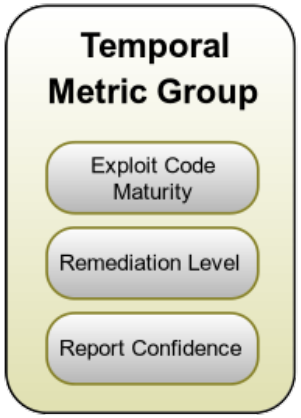
– CVSS Scores & Vulnerability Types

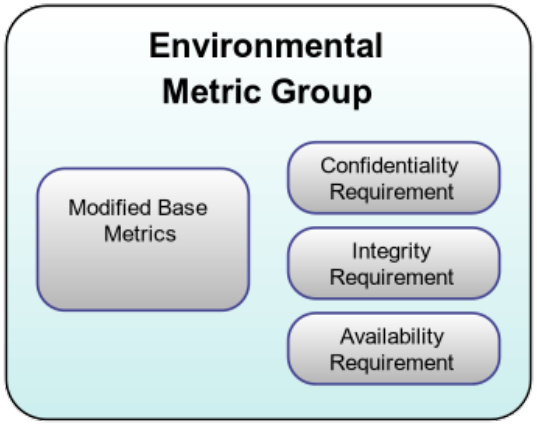
CVSS Score	7.2
Confidentiality Impact	Complete (There is total information disclosure, resulting in all system files being revealed.)
Integrity Impact	Complete (There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised.)
Availability Impact	Complete (There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable.)
Access Complexity	Low (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit.)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	None
Vulnerability Type(s)	Execute Code
CWE ID	264

CVSS Version3 Scoring System Calculator

- Es gibt Webseiten zur Berechnung vom CVSS Score mit ausführlicher Dokumentation
  - https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator
  - https://www.first.org/cvss/calculator/3.1
- Folgende Metriken gehen in den score ein







https://www.first.org/cvss/specification-document

OWASP Dependency-Check

- https://owasp.org/www-project-dependency-check/
- Automatisiertes Tool zum Analysieren der eingesetzten Libraries
- Wertet die CVEs der National Vulnerability Database aus
  - Siehe http://nvd.nist.gov/
  - Einträge werden im folgenden Format gespeichert:

```
<entry id="CVE-2014-3578">
...
<vuln:vulnerable-software-list>
          <vuln:product>cpe:/a:pivotal:spring_framework:4.0.4</vuln:product>
          <vuln:product>cpe:/a:pivotal:spring_framework:4.0.3</vuln:product>
          ...
          <vuln:product>cpe:/a:pivotal:spring_framework:3.0.4</vuln:product>
<vuln:vulnerable-software-list>
...
</entry>
```

- Vendor, Product und Version werden aus pom.xml bzw. der Manifest-Datei bezogen

OWASP Dependency-Check

- Scan-Optionen
  - Command Line Tool
  - Maven Plugin
  - Ant Task
  - Gradle Plugin
  - Jenkins Plugin

- Wie bei allen automatisierten Tools:
  Man muss False Positives und False Negatives beachten
  - False Positives können durch Konfiguration aus dem Scan entfernt werden
  - Beachtet vor allem Vorkommen mit 0 findings
  - Hier lohnt sich eine manuelle Recherche

Was tun bei gefundenen Sicherheitslücken?

- Schritt 1 – Informationssammlung:
  - Welche Komponente ist betroffen, Was ist der Angriffsvektor, Was sind die Auswirkungen, Gibt es Updates / Workarounds, Gibt es schon Exploits
  - CVE oft guter Ausgangspunkt für weitere Recherchen
- Schritt 2 – Evaluation:
  - Wie wirkt sich die Sicherheitslücke auf meine Systemsicherheit aus
  - Was sind geeignete Maßnahmen, um Verwundbarkeit des Systems zu vermeiden
    - Update einspielen, Bibliothek austauschen durch Alternative
    - Oft nützlich: kurzfristige und langfristige Maßnahmen definieren
- Schritt 3 – Maßnahmen umsetzen

Zusammenfassung Secure Software Engineering

- Das Thema Sicherheit muss von Anfang an in den Software Engineering Prozess integriert werden.
- Das Thema Sicherheit muss kontinuierlich auf dem Software-Fließband gemessen werden