

# IT-Sicherheit



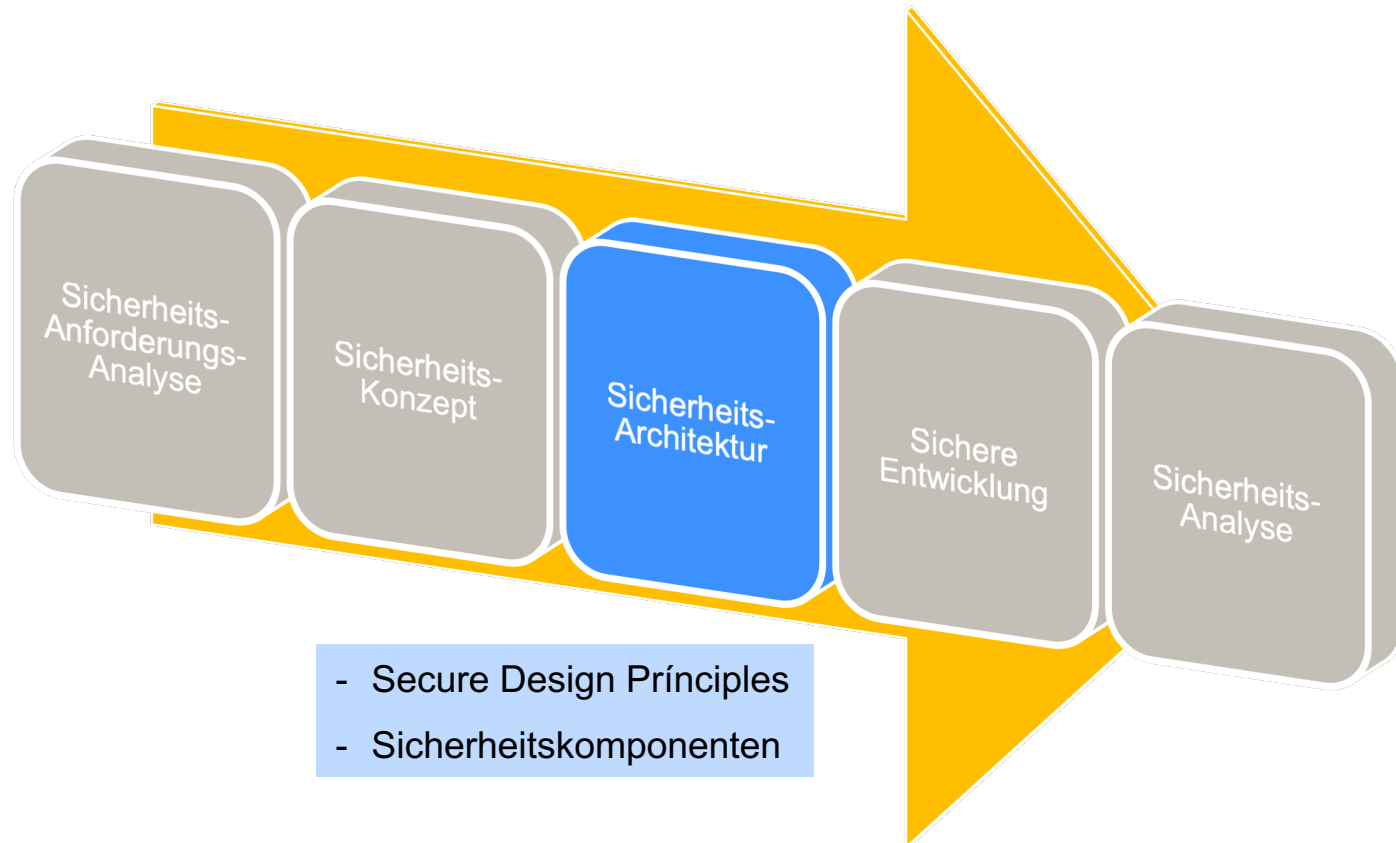
## Kapitel 6: Secure Software Engineering Teil 2

- ▶ Sicherheitsarchitektur und- Design
- ▶ Tools zur Sicherheitsanalyse





# Sicherheitsarchitektur und -Design





# 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>



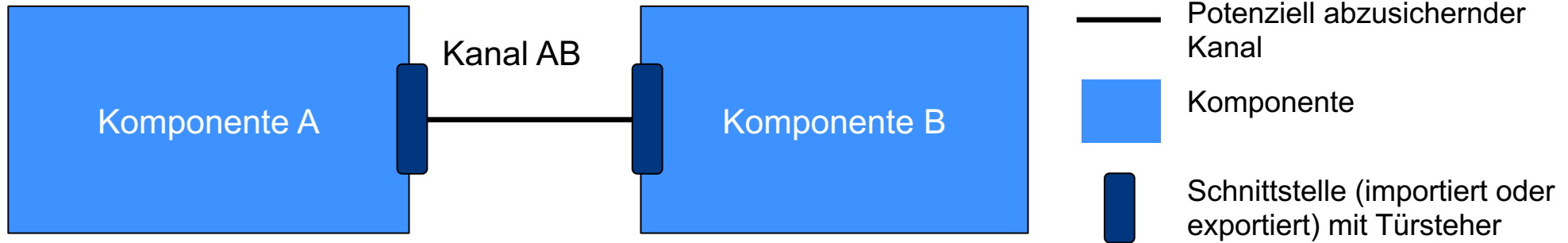
# Secure Design Principles

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	<ul style="list-style-type: none"> <li>• HTTPS mit Perfect Forward Secrecy</li> <li>• Verschlüsselung auf Anwendungsebene</li> </ul>
-- " --	Prüfen, ob der Absender die Nachricht auch senden darf	<ul style="list-style-type: none"> <li>• URLs gegen ACL prüfen (.htaccess)</li> <li>• Anwendungsebene: Inhalt der Nachricht</li> </ul>
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 nutzen

- ▶ **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

- ▶ **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

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





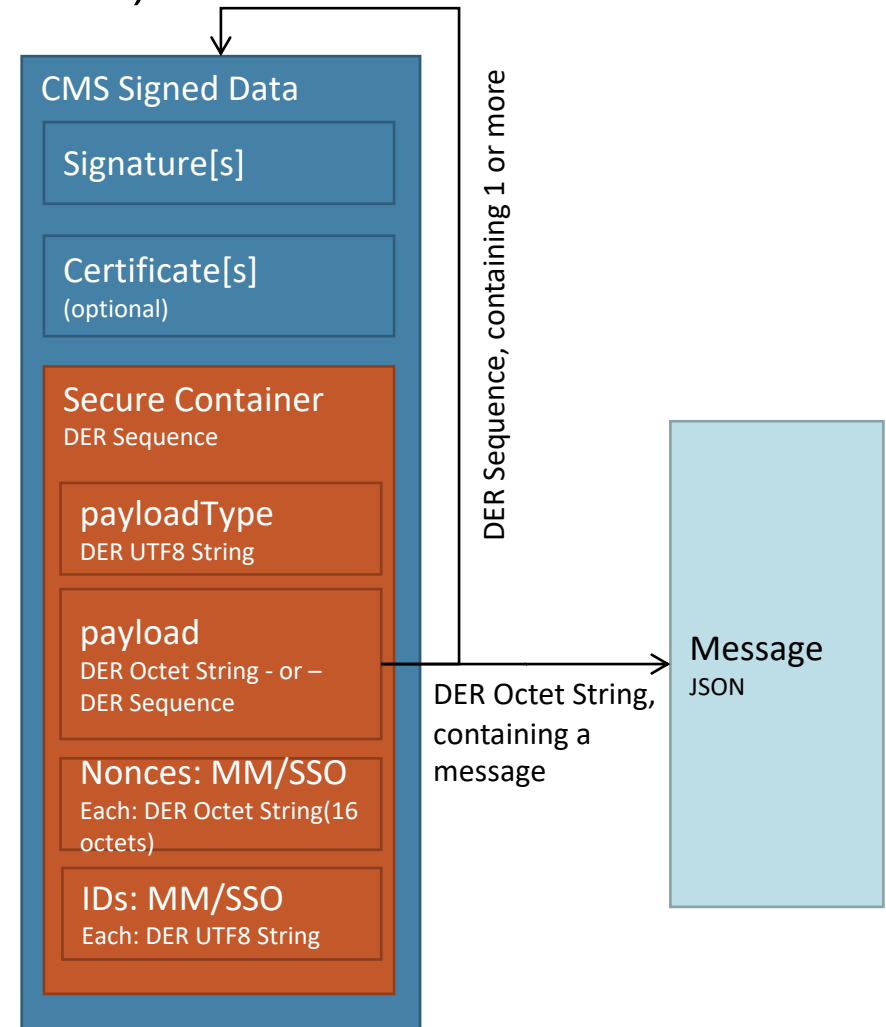
## Die restlichen Prüfungen sind von fachlicher Natur

- ▶ 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 angeben
  - ▶ 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

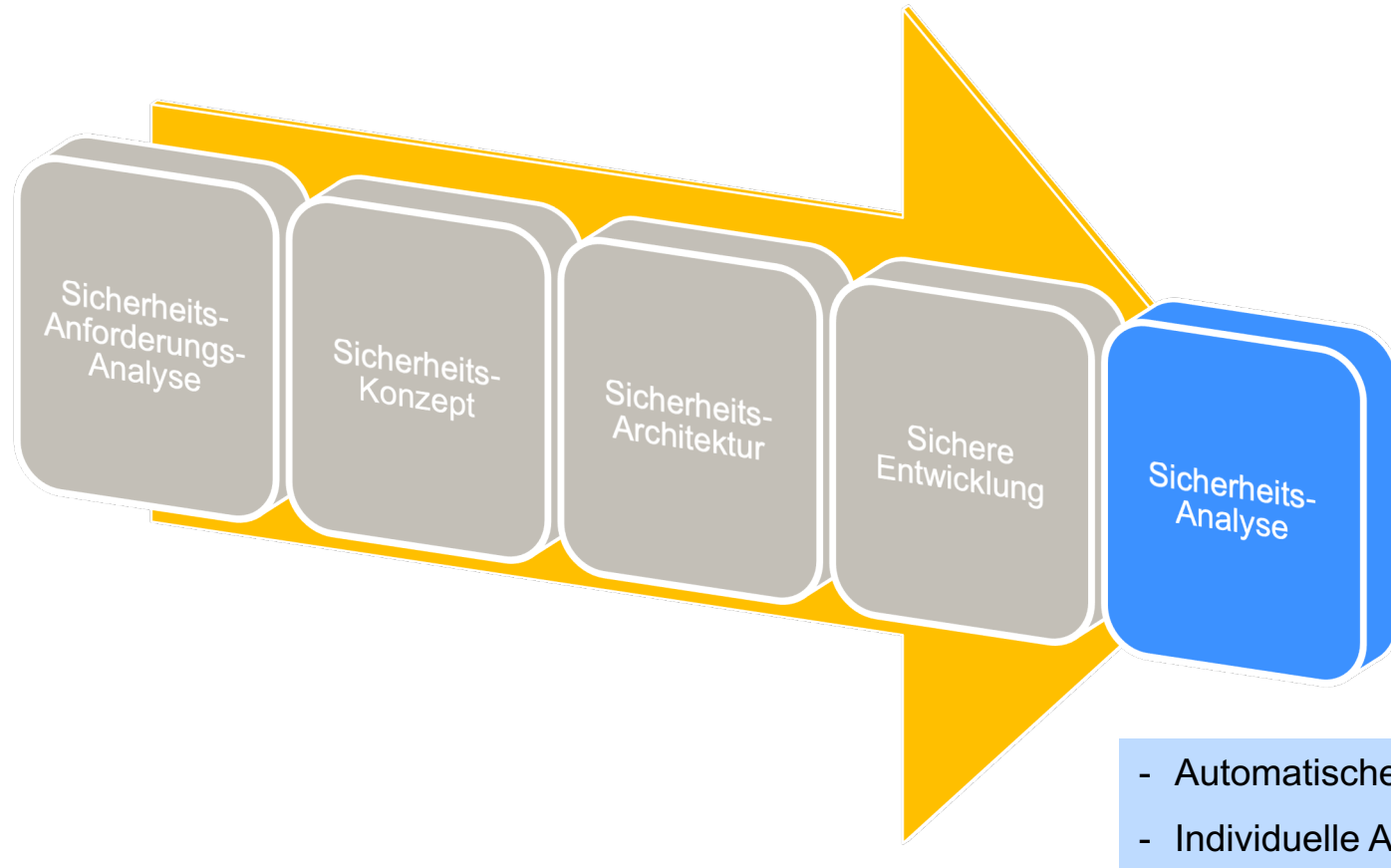
### ▶ CMS Cryptographic Message Syntax (PKCS#7)

- ▶ Nachrichten signieren
- ▶ Nachrichten verschlüsseln
- ▶ Der Standard, kann mit den bekannten Bibliotheken geprüft werden, z.B. OpenSSL





# Sicherheitsanalyse





# 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

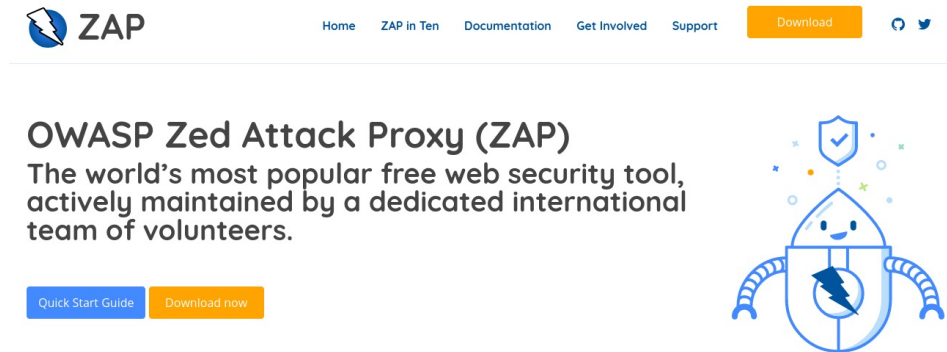
- ▶ Gegenseitige Reviews sind zusätzlich zu SAST notwendig



# 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

Header: Rohdaten anzeigen Body: Rohdaten anzeigen

```
GET http://localhost:8080/ToDo/create HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Referer: http://localhost:8080/ToDo/search?q=murf
Cookie: JSESSIONID=95EBC728F4FAACA44C728FC2556CD45A
Connection: keep-alive
Host: localhost:8080
```

Id	Req. Timestamp	Metho...	URL	Co...	Reason	R...	Size Resp. Bo...	Highest Al...	No...	Tags
8	24.02.15 23:17:...	GET	http://localhost:8080/ToDo/img?src=qawa...	200	OK	6...	5,91 KiB	Niedrig		
9	24.02.15 23:18:...	POST	http://localhost:8080/ToDo/create	302	Found	1...	0 Byte			
10	24.02.15 23:18:...	GET	http://localhost:8080/ToDo/	200	OK	2...	1,23 KiB	Niedrig		Form
11	24.02.15 23:18:...	GET	http://localhost:8080/ToDo/img?src=qawa...	200	OK	5...	5,91 KiB	Niedrig		
12	24.02.15 23:22:...	GET	http://localhost:8080/ToDo/register	200	OK	8...	663 Byte	Niedrig		Form
13	24.02.15 23:23:...	GET	http://localhost:8080/ToDo/img?src=qawa...	200	OK	5...	5,91 KiB	Niedrig		
14	24.02.15 23:23:...	POST	http://localhost:8080/ToDo/register	200	OK	5...	720 Byte	Niedrig		Form
15	24.02.15 23:23:...	GET	http://localhost:8080/ToDo/img?src=qawa...	200	OK	1...	5,91 KiB	Niedrig		

Warnungen 4 0 4 1 Laufende Scans 0 0 0 0 0 0





# 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

Id	Req. Timestamp	Metho...	URL	Co...	Reason	R...	Size Resp. Bo...	Highest Al...	No...	Tags
8	24.02.15 23:17:...	GET	http://localhost:8080/./img?src=qawa...	200	OK	6 ...	5,91 KiB	Niedrig		
9	24.02.15 23:18:...	POST	http://localhost:8080/./create	302	Found	1...	0 Byte			
10	24.02.15 23:18:...	GET	http://localhost:8080/./	200	OK	2...	1,23 KiB	Niedrig		Form
11	24.02.15 23:18:...	GET	http://localhost:8080/./img?src=qawa...	200	OK	5 ...	5,91 KiB	Niedrig		
12	24.02.15 23:22:...	GET	http://localhost:8080/./register	200	OK	8...	663 Byte	Niedrig		Form
13	24.02.15 23:23:...	GET	http://localhost:8080/./img?src=qawa...	200	OK	5 ...	5,91 KiB	Niedrig		
14	24.02.15 23:23:...	POST	http://localhost:8080/./register	200	OK	5...	720 Byte	Niedrig		Form
15	24.02.15 23:23:...	GET	http://localhost:8080/./img?src=qawa...	200	OK	1...	5,91 KiB	Niedrig		





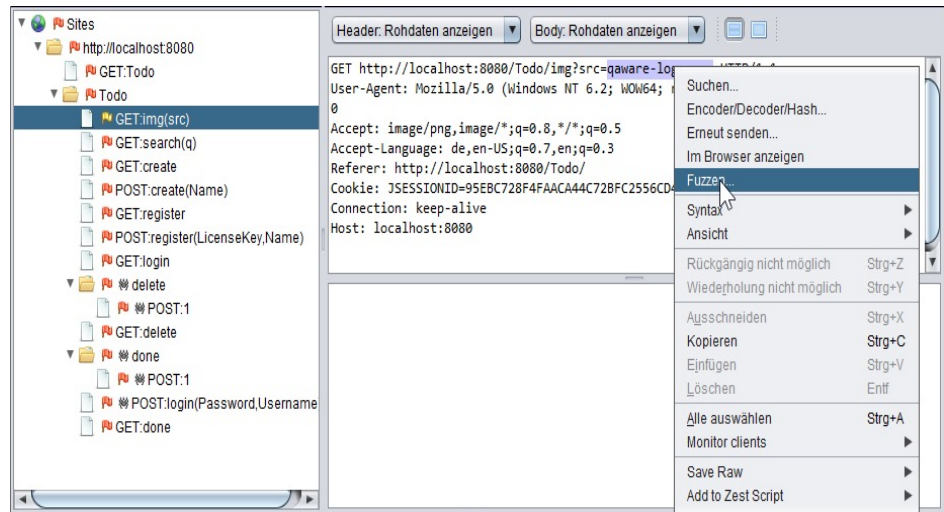
# OWASP ZAP - ZAP Attack Proxy



## Features II:

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

Id	Req. Timesta...	Resp. Timesta...	Meth...	URL	Co...	Reason	R...	Size Resp. He...	Size Resp. ....
665	24.02.15 23:3...	24.02.15 23:3...	POST	http://localhost:8080/Todo/create	200	OK	3...	149 Byte	27,84 KiB
666	24.02.15 23:3...	24.02.15 23:3...	POST	http://localhost:8080/Todo/delete/1	200	OK	2...	232 Byte	27,84 KiB
667	24.02.15 23:3...	24.02.15 23:3...	POST	http://localhost:8080/Todo/delete/1	200	OK	4...	232 Byte	27,84 KiB
668	24.02.15 23:3...	24.02.15 23:3...	GET	http://localhost:8080/Todo/done	404	Not Fo...	1...	172 Byte	949 Byte
669	24.02.15 23:3...	24.02.15 23:3...	GET	http://localhost:8080/Todo/done	404	Not Fo...	4...	172 Byte	949 Byte
670	24.02.15 23:3...	24.02.15 23:3...	GET	http://localhost:8080/Todo/done	505	HTTP V...	3...	126 Byte	0 Byte
671	24.02.15 23:3...	24.02.15 23:3...	POST	http://localhost:8080/Todo/done/1	200	OK	2...	232 Byte	27,84 KiB
672	24.02.15 23:3...	24.02.15 23:3...	POST	http://localhost:8080/Todo/done/1	200	OK	4...	232 Byte	27,84 KiB





# 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 Ausspähöns 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 Informationstechnologie, 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

### 699 - Software Development

- + **C** API / Function Errors - (1228)
- + **C** Audit / Logging Errors - (1210)
  - **B** Improper Output Neutralization for Logs - (117)
  - **B** Truncation of Security-relevant Information - (223)
  - **B** Omission of Security-relevant Information - (223)
  - **B** Obscured Security-relevant Information by Alter
  - **B** Insertion of Sensitive Information into Log File -
  - **B** Insufficient Logging - (778)
  - **B** Logging of Excessive Data - (779)
- + **C** Authentication Errors - (1211)
- + **C** Authorization Errors - (1212)
- + **C** Bad Coding Practices - (1006)
- + **C** Behavioral Problems - (438)
- + **C** Business Logic Errors - (840)
- + **C** Communication Channel Errors - (417)
- + **C** Complexity Issues - (1226)
- + **C** Concurrency Issues - (557)
- + **C** Credentials Management Errors - (255)
- + **C** Cryptographic Issues - (310)
- + **C** Data Integrity Issues - (1214)
- + **C** Data Processing Errors - (19)
- + **C** Data Representation Errors - (137)
- + **C** Documentation Issues - (1225)
- + **C** File Handling Issues - (1219)

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





# The CWE – Top 25 2019

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

[https://cwe.mitre.org/top25/archive/2019/2019\\_cwe\\_top25.html](https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html)

## ▶ 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

### CVE Details

The ultimate security vulnerability datasource

[Log In](#) [Register](#)

[Home](#)

**Browse :**

[Vendors](#)

[Products](#)

[Vulnerabilities By Date](#)

[Vulnerabilities By Type](#)

**Reports :**

[CVSS Score Report](#)

[CVSS Score Distribution](#)

**Search :**

[Vendor Search](#)

[Product Search](#)

[Version Search](#)

[Vulnerability Search](#)

[By Microsoft References](#)

**Top 50 :**

[Vendors](#)

[Vendor Cvss Scores](#)

[Products](#)

[Product Cvss Scores](#)

[Versions](#)

**Other :**

[Microsoft Bulletins](#)

[Bugtraq Entries](#)

[CVE Definitions](#)

[About & Contact](#)

[Feedback](#)

[CVE Help](#)

[FAQ](#)

[Articles](#)

**External Links :**

[NVD Website](#)

[CVE Web Site](#)

**View CVE :**

(e.g.: CVE-2009-1234 or 2010-1234 or 20101234)

**View BID :**

#### Vulnerability Details : [CVE-2019-0211](#)

In Apache HTTP Server 2.4 releases 2.4.17 to 2.4.38, with MPM event, worker or prefork, code executing in less-privileged process scripting interpreter) could execute arbitrary code with the privileges of the parent process (usually root) by

Publish Date : 2019-04-08 Last Update Date : 2019-06-11

[Collapse All](#) [Expand All](#) [Select](#) [Select&Copy](#) [Scroll To](#) [Comments](#) [External Links](#)

[Search Twitter](#) [Search YouTube](#) [Search Google](#)

#### – CVSS Scores & Vulnerability Types

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

#### – Related OVAL Definitions

Title	Definition Id	Class	Family
RHSA-2019:0980: httpd:2.4 security update (Important)	<a href="#">oval:com.redhat.rhsa:def:20190980</a>		unix

OVAL (Open Vulnerability and Assessment Language) definitions define exactly what should be done to verify a vulnerability or a mit to verify a vulnerability.

#### – Products Affected By CVE-2019-0211

#	Product Type	Vendor	Product	Version	Update	Edition	Language
1	Application	<a href="#">Apache</a>	<a href="#">Http Server</a>	2.4.17			<a href="#">Version Details</a> <a href="#">Vulnerabilities</a>
2	Application	<a href="#">Apache</a>	<a href="#">Http Server</a>	2.4.18			<a href="#">Version Details</a> <a href="#">Vulnerabilities</a>
3	Application	<a href="#">Apache</a>	<a href="#">Http Server</a>	2.4.19			<a href="#">Version Details</a> <a href="#">Vulnerabilities</a>
4	Application	<a href="#">Apache</a>	<a href="#">Http Server</a>	2.4.20			<a href="#">Version Details</a> <a href="#">Vulnerabilities</a>

<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

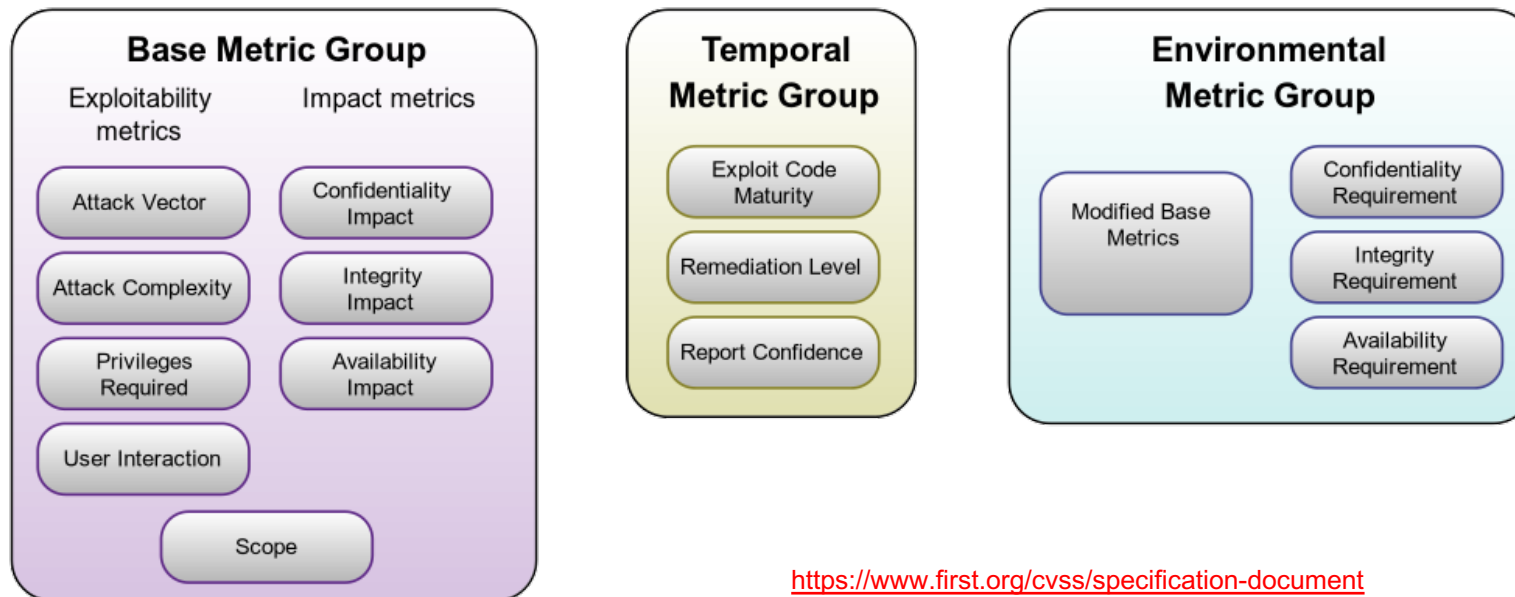
CVSS Score	<b>7.2</b>
Confidentiality Impact	<b>Complete</b> (There is total information disclosure, resulting in all system files being revealed.)
Integrity Impact	<b>Complete</b> (There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire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

# ▶ 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





# 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

