



CAS Enterprise Application Development Java EE

Modul Java EE Security

- ▶ Peter Andres – ISC-EJPD

Zu meiner Person



Name, Vorname:

Kontaktangaben:

Peter Andres

ISC-EJPD

Güterstrasse 24, 3003 Bern

✉ peter.andres@isc-ejpd.admin.ch

☎ +41 (0)58 46 37872

Funktion beim ISC-EJPD

Abteilungsleiter Technologie (CTO)

Letzte Ausbildungen

CAS Information Security (fhnw)
Certified Information Systems Security
Professional (CISSP)
CAS Security & Privacy (bfh)
MAS-IT (bfh)

Erwartungen an den Kurs



Lernziel

Ziel dieses Kurses ist es, die grundlegenden Sicherheitstechniken, welche die JEE 7 Plattform bietet, zu verstehen und in der Praxis anwenden zu können

Lerninhalte

1

Einleitung
Überblick

2

Grundlagen
Einfache Beispiel

3

Grundlagen
Begriffe

4

Sicherheit im
Web-Tier

5

Sicherheit im
Web-Tier (Advanced)

6

Sicherheit im
EJB-Tier

7

Sicherheit
Webservices

8

Ausblick JEE8
(Sicherheit)

9

Transportsicherheit
Secure Sockets Layer

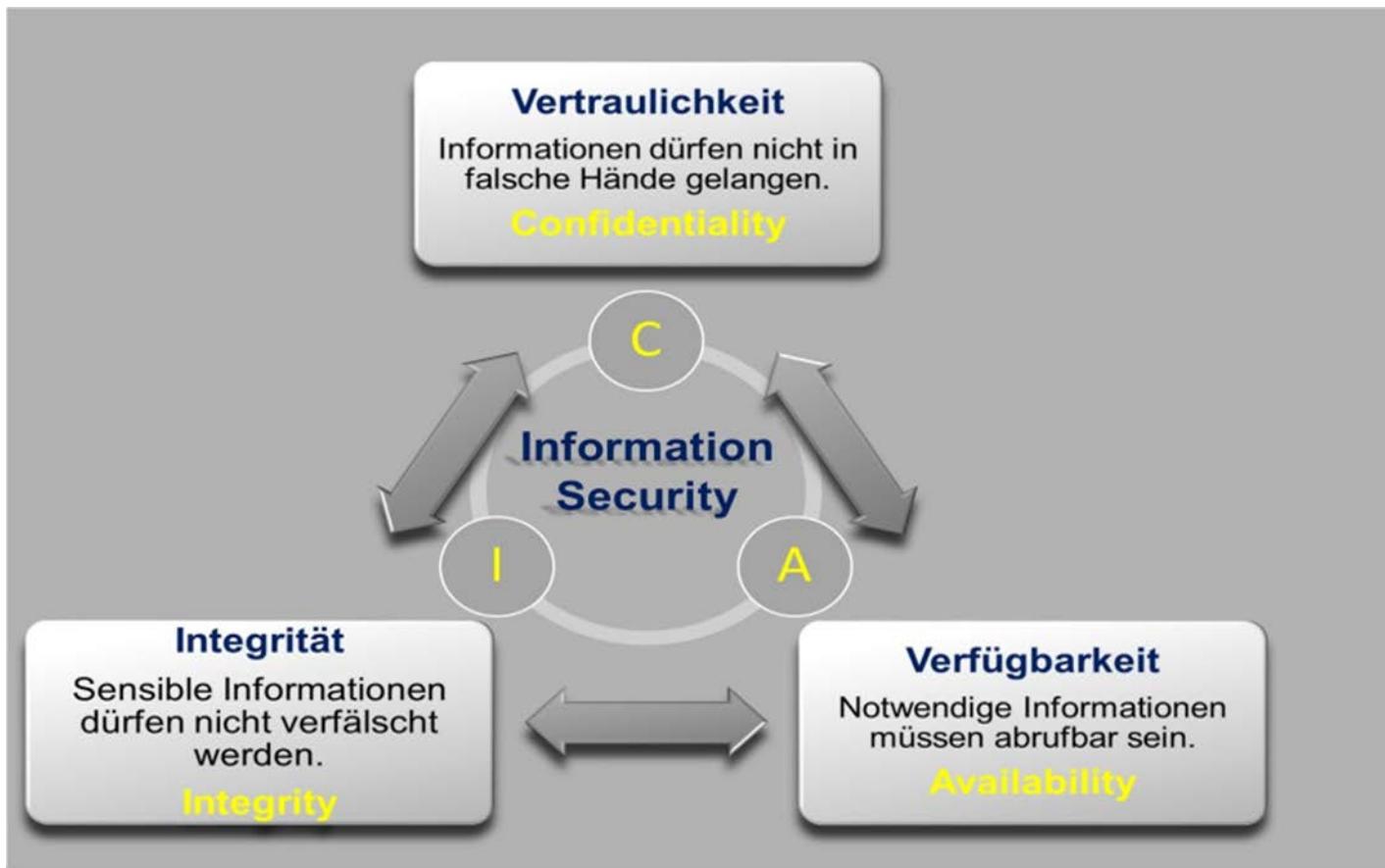
10

Übungen
Bookstore

Einleitung

Überblick und Einführung

Java EE 7 Security Einleitung: C.I.A. Prinzip / Dreieck



Quelle comupterwoche

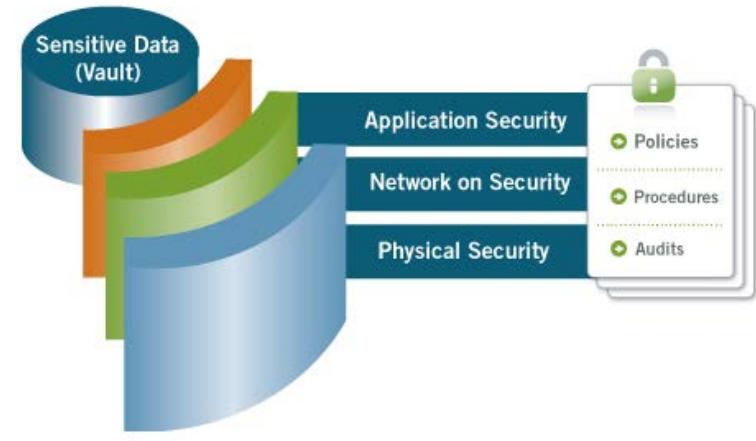
Java EE 7 Security Einleitung: Integrale Sicherheit



Java EE 7 Security Einleitung: Defense in depth



Quelle nsslabs



Quelle nsslabs

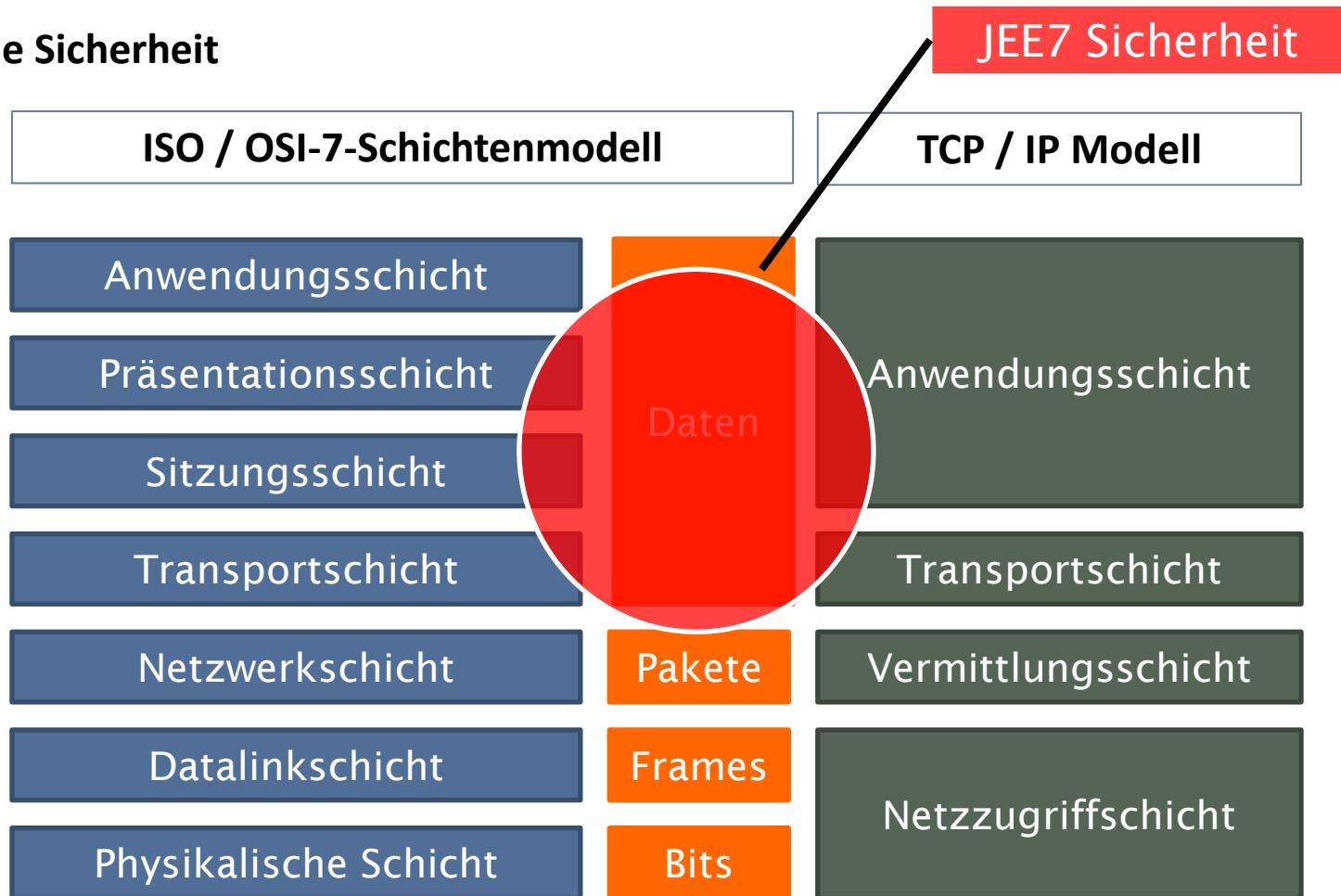
Java EE 7 Security Einleitung: Kleines Beispiel

Wie sicher ist eine ...

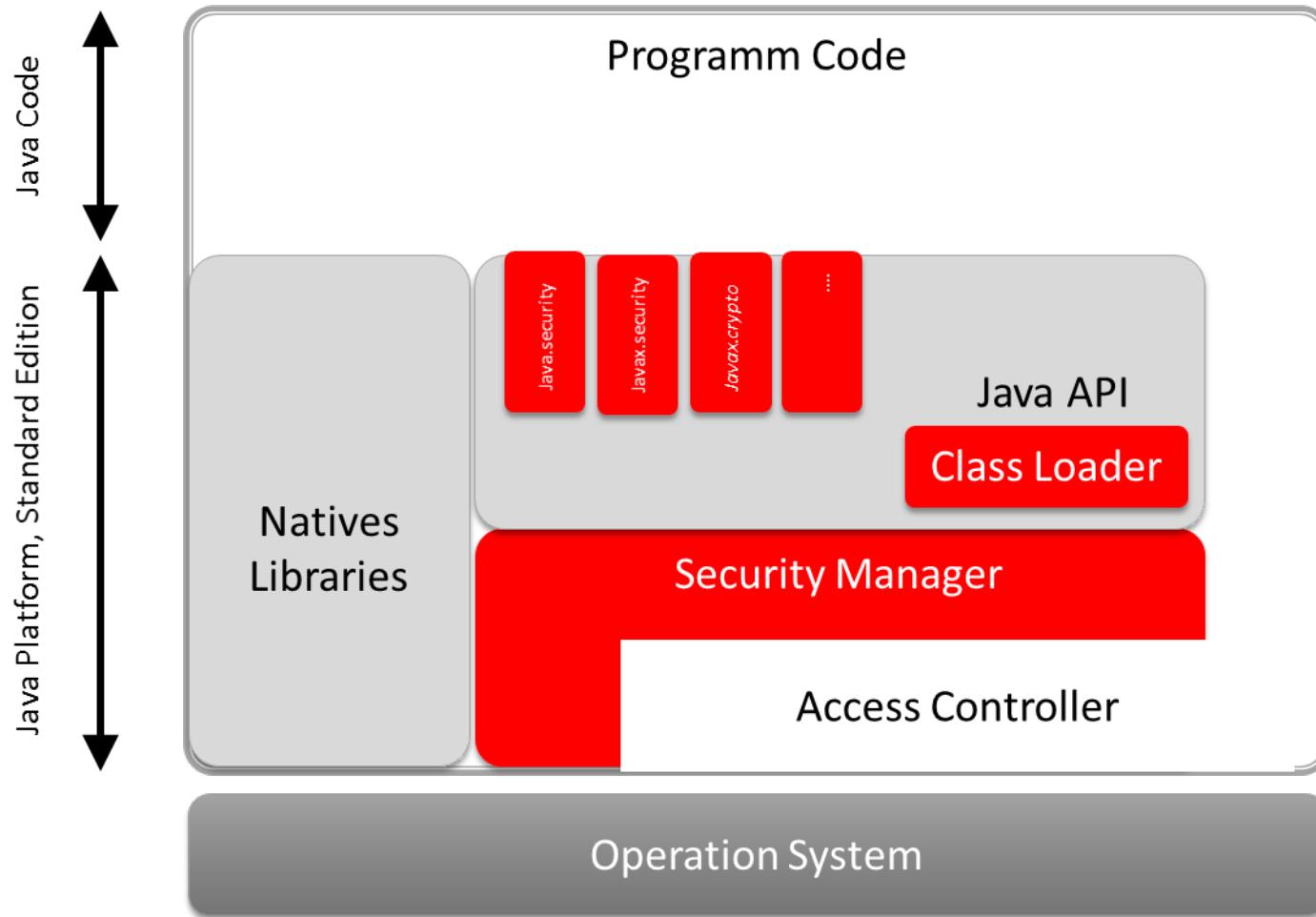


Java EE 7 Security Einleitung: IT-Sicherheit, OSI Modell

Technische Sicherheit



Java EE 7 Security Einleitung: Java SE Security



Java EE 7 Security Einleitung: Java 7 SE Security: Links

- ▶ **Allgemeines über Java SE Security**
 - <http://docs.oracle.com/javase/7/docs/technotes/guides/security/>
 - <http://docs.oracle.com/javase/tutorial/essential/environment/security.html>
- ▶ **Java Security-Manager (gute Übersicht, Fachhochschule Nordwestschweiz)**
 - <http://grunz.ch/courses/sem/ss03/JavaSecurityManager.pdf>
- ▶ **Java Classloader (gute Übersicht, Fachhochschule Nordwestschweiz)**
 - <http://www.grunz.ch/courses/sem/ws03/ClassLoaders.pdf>

Java EE 7 Security Einleitung: Java 7 SE Security: Links

► **JCE (Java Cryptography Extension)**

- JCE ist ein Java-API und Framework für kryptographische Aufgaben wie Verschlüsselung, Message-Digest, Authentisierung und Schlüsselverwaltung.
- <http://docs.oracle.com/javase/7/docs/technotes/guides/security/>

► **JSSE (Java Secure Socket Extension)**

- Java JSSE ist ein Java-API für Secure Sockets Layer (SSL).
- <https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html>

► **SASL (Simple Authentication and Security Layer)**

- SASL ist ein Framework, das von verschiedenen Protokollen zur Authentisierung verwendet werden kann
- <https://docs.oracle.com/javase/7/docs/technotes/guides/security/sasl/sasl-refguide.html>

Java EE 7 Security Einleitung: OWASP Top 10

JEE Security

Applikation-Security

OWASP Top 10 – 2010 (alt)	Δ	OWASP Top 10 – 2013 (neu)
A1 – Injection	=	A1 – Injection
A3 – Fehler in Authentifizierung und Session-Management	↗	A2 – Fehler in Authentifizierung und Session-Management
A2 – Cross-Site Scripting (XSS)	↘	A3 – Cross-Site Scripting (XSS)
A4 – Unsichere direkte Objektreferenzen	=	A4 – Unsichere direkte Objektreferenzen
A6 – Sicherheitsrelevante Fehlkonfiguration	↗	A5 – Sicherheitsrelevante Fehlkonfiguration
A7 – Kryptografisch unsichere Speicherung – mit A9 →	↗	A6 – Verlust der Vertraulichkeit sensibler Daten
A8 – Mangelhafter URL-Zugriffsschutz – erweitert zu →	↗	A7 – Fehlerhafte Autorisierung auf Anwendungsebene
A5 – Cross-Site Request Forgery (CSRF)	↘	A8 – Cross-Site Request Forgery (CSRF)
<Teil von A6: Sicherheitsrelevante Fehlkonfiguration>	neu	A9 – Verwendung von Komponenten mit bekannten Schwachstellen
A10 – Ungeprüfte Um- und Weiterleitungen	=	A10 – Ungeprüfte Um- und Weiterleitungen
A9 – Unzureichende Absicherung der Transportschicht	↗	Zusammen mit 2010-A7 nun im neuen 2013-A6

Java EE 7 Security Einleitung: Heartbleed Lücke ...

Einfach erklärt

<http://www.heise.de/security/artikel/So-funktioniert-der-Heartbleed-Exploit-2168010.html>

Demo

<https://www.youtube.com/watch?v=jaobW2hd6Ho>

<https://www.youtube.com/watch?v=OMtvF-FTxGQ>

<https://www.youtube.com/watch?v=hTK0pywfmDE>

2

Grundlagen Einfaches Beispiel

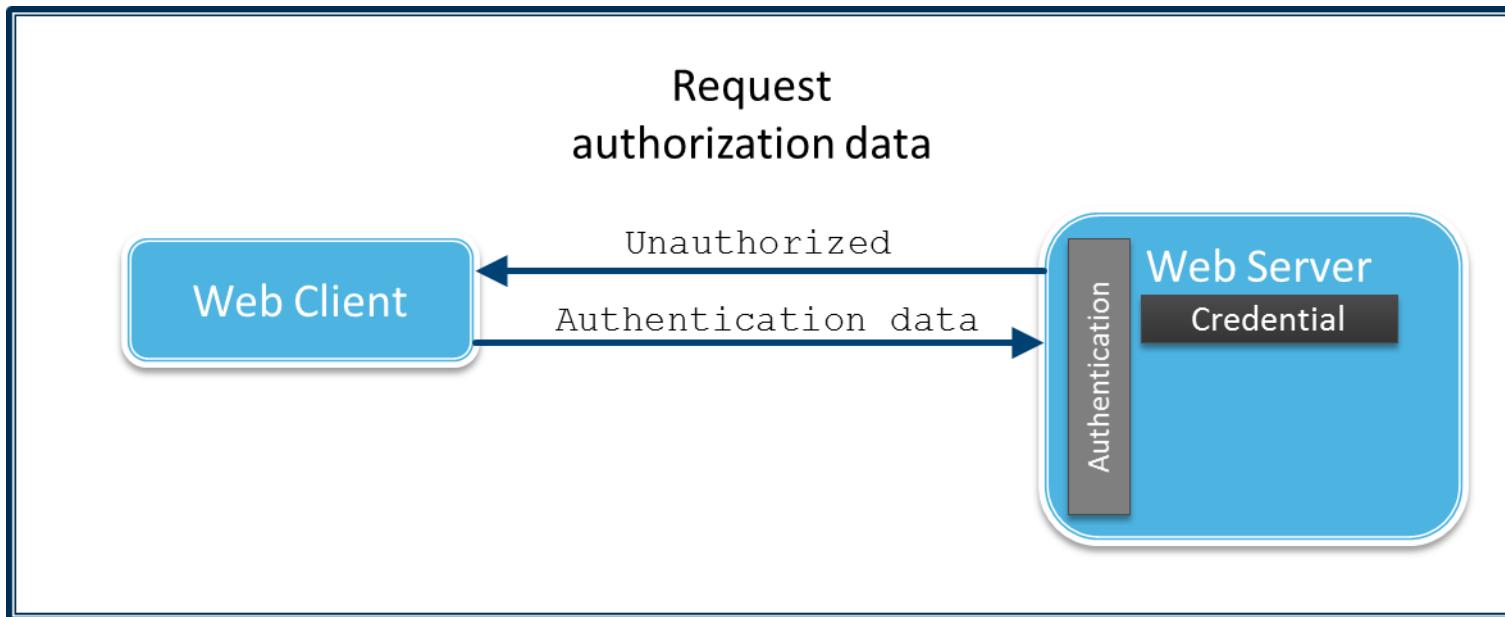
Java EE 7 Security: Ein einfaches Beispiel, Schritt 1

Initialer (erstmaliger) Request



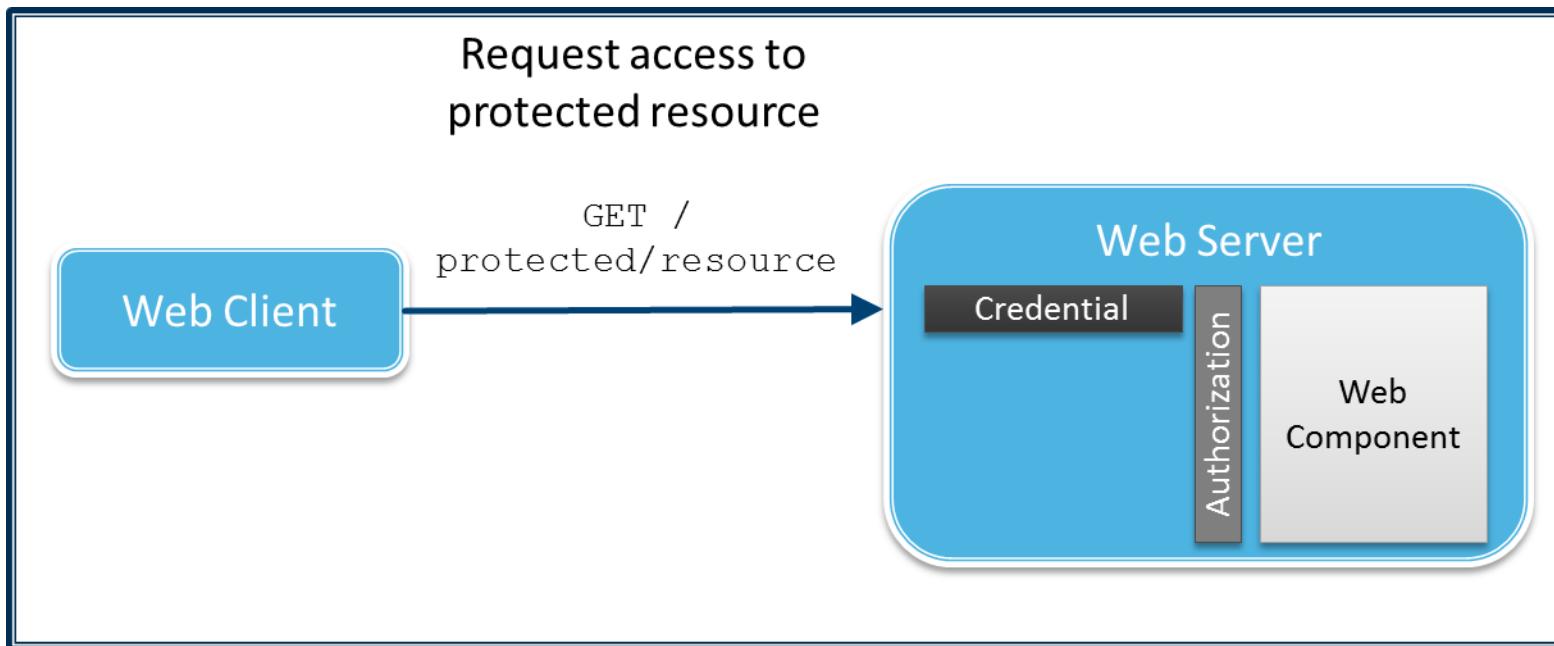
Java EE 7 Security: Ein einfaches Beispiel, Schritt 2

Initiale Authentisierung des Web Client (Client)



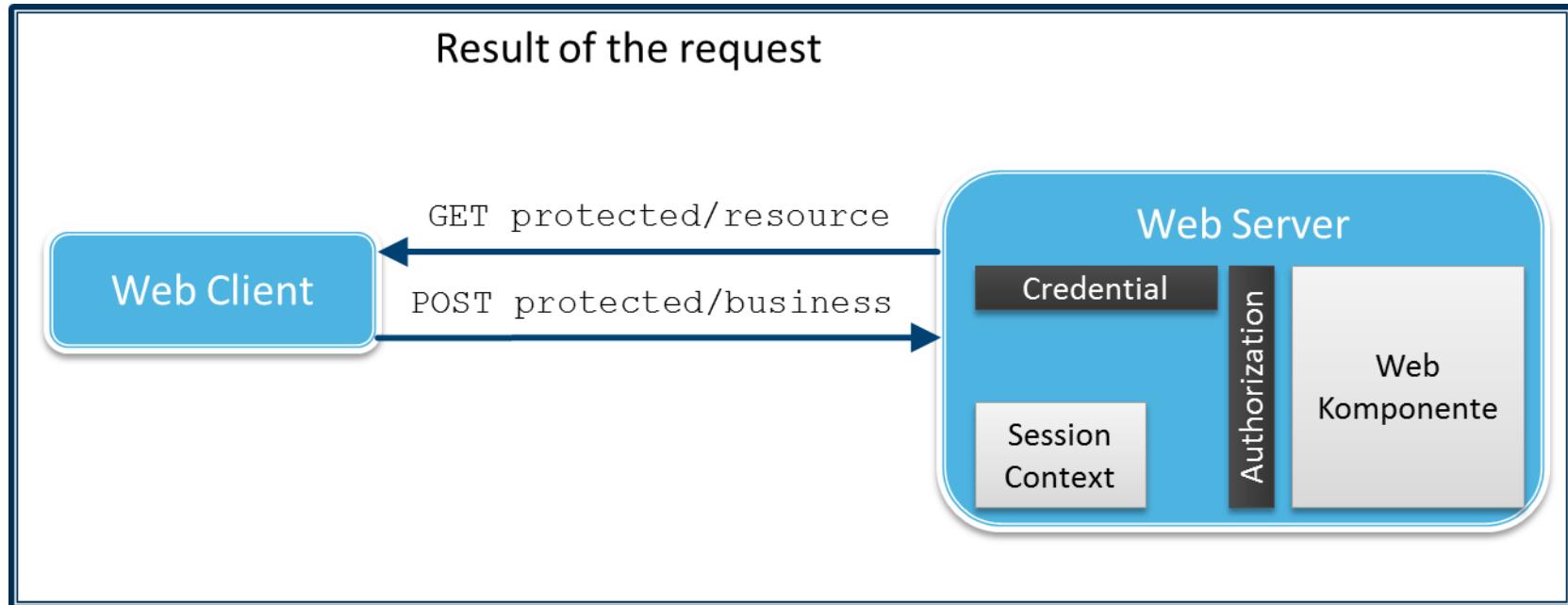
Java EE 7 Security: Ein einfaches Beispiel, Schritt 3

Autorisierung des Web Client



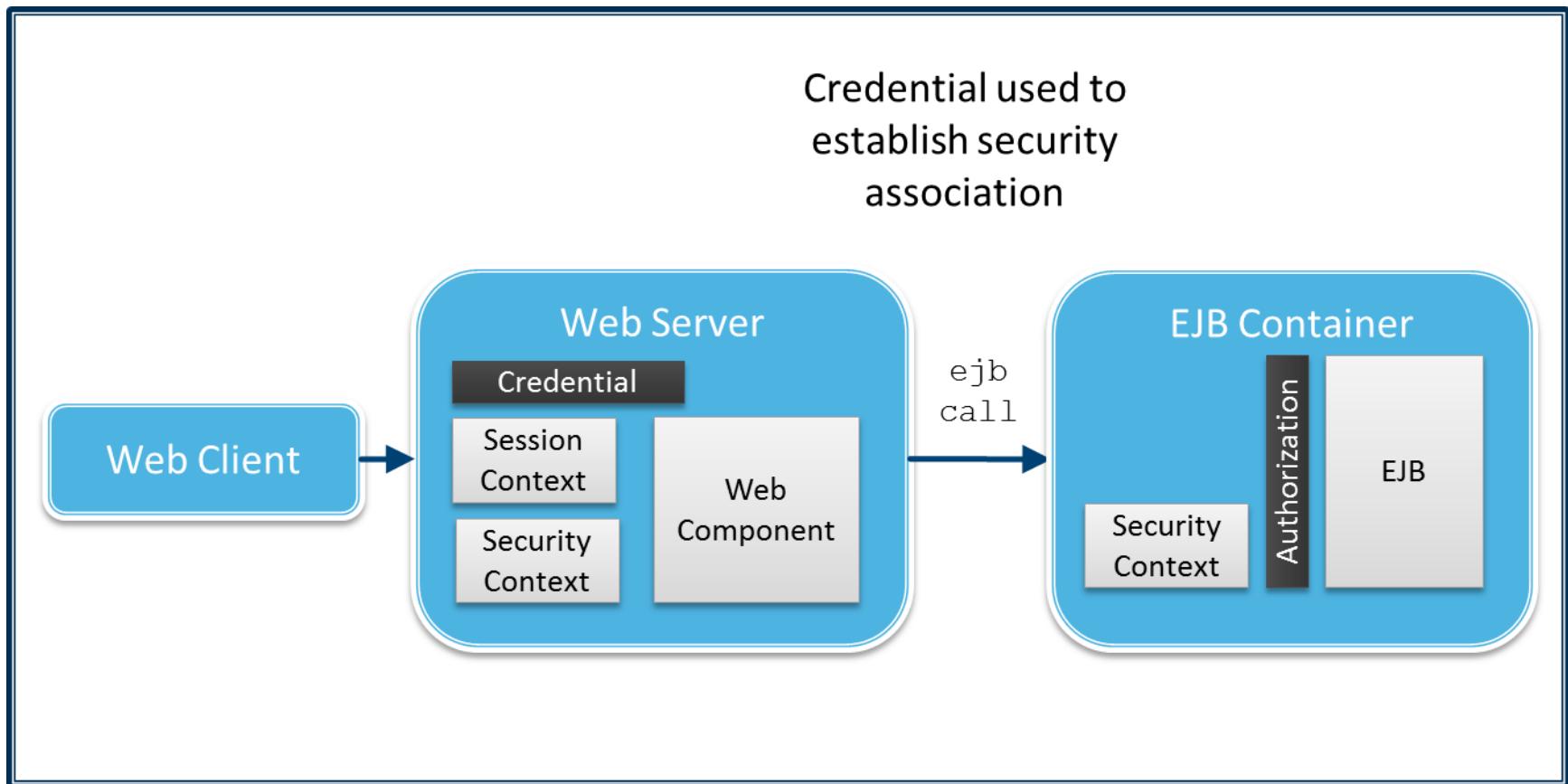
Java EE 7 Security: Ein einfaches Beispiel, Schritt 4

Geschützte Ressource ausliefern, Business-Call ausführen



Java EE 7 Security: Ein einfaches Beispiel, Schritt 5

Businesscall ausführen



Java EE 7 Security: Ein einfaches Beispiel, Demo

In der Praxis sieht das dann folgendermassen aus ...



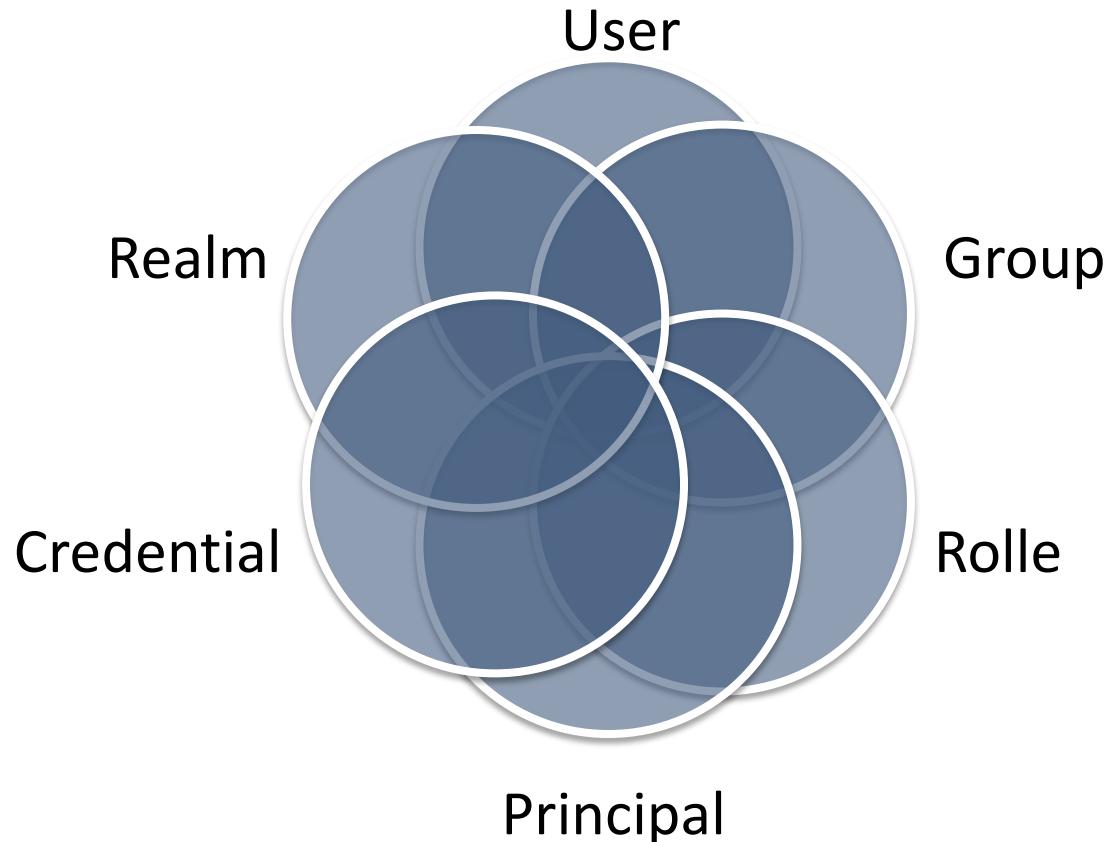
Java EE 7 Security : Das Ziel



3

Grundlagen Begriffserklärung

Java EE 7 Security – Grundlagen: Begriffserklärung



Java EE 7 Security – Grundlagen: Begriffserklärung

User

Ein Benutzer ist eine Identität, welche im Identity-Store (z.B. LDP, Datei usw.) angelegt und definiert wurde. Ein Benutzer kann eine oder mehrere Rollen einnehmen und kann einer Gruppe zugeordnet werden.

Group

Eine Gruppe besteht aus einem Set von Usern, mit gemeinsamen Merkmalen, die in der Regel über eine Reihe von gemeinsamen Berechtigungen verfügen. Eine Gruppe kann eine oder mehrere Rollen einnehmen.

Rolle

Eine Rolle ist ein abstrakter Name für eine Berechtigung einer geschützten Ressource innerhalb einer Applikation. Jede Rolle kann einem oder mehreren Benutzern und / oder Gruppen zugewiesen werden.

Java EE 7 Security – Grundlagen: Begriffserklärung

Credential

Ein Credential ist ein Identitätsnachweis, der einem System die Identität eines Benutzers (oder eines Systems) bestätigt. Dies können Ausweispapiere, Passwörter, Ergebnisse von kryptografischen Verfahren usw. sein.

Principal

Ein Principal ist eine Identität, die anhand eines bekannten Credential authentisiert werden kann.

Java EE 7 Security – Grundlagen: Begriffserklärung

Security Realm

Ein (Security) Realm ist eine definierte Sicherheits-Domäne innerhalb eines Web- oder Applikationsservers. Zu dieser Domäne können verschiedene Benutzer und Gruppen von Benutzern aus unterschiedlichen Identity-Stores (z.B. LDAP) hinterlegt werden.

Jede abgesicherte Applikation wird genau mit einem solchem Realm verlinkt und holt sich über diesen die Benutzerdaten zur Validierung.

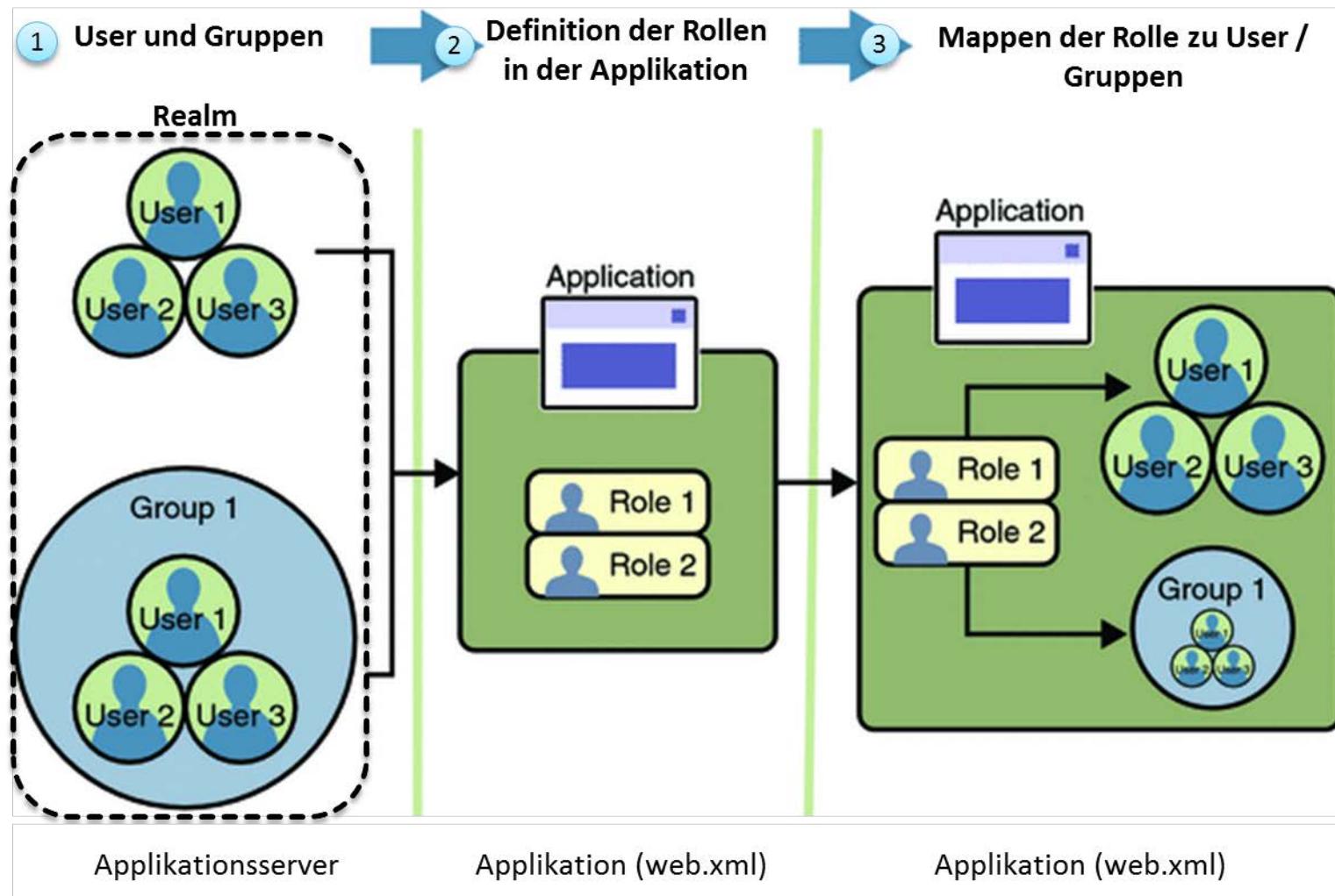
Die Web- und Applikationsserver bieten meist verschiedene Möglichkeiten zur Definition (Dateien, Datenbanken, LDAP usw.). So ist es z.B. fast immer möglich, Benutzer und Gruppen mittels Key-Value-Dateien zu definieren.

Java EE 7 Security – Grundlagen: Basis – «Rollen»

Rollen-Definition gemäss Oracle (im JEE7 Kontext)

- ▶ Der Applikationsentwickler definiert, wie User authentisiert werden sollen (Definition der LOA – Level of Assurance, z.B. mit Username / Passwort)
- ▶ Der Applikationsentwickler definiert welche Ressourcen der Applikation zu schützen sind (mit Hilfe von Annotationen, Deskriptoren oder programmatisch)
- ▶ Der Administrator erstellt im Applicationserver (oder in einem Identity-Store) Benutzer und Gruppen
- ▶ Der Applikationsdeployer ordnet Rollen den Usern und / oder Gruppen zu. Der Applicationserver mapped Rollen zu den Usern und Gruppen

Java EE 7 Security – Grundlagen: Benutzer Mapping



Java EE 7 Security – Grundlagen: Praxisbezug

Hier am Beispiel des Glassfish-Application-Server 4.1

«File-based» Realm

Benutzer: admin | Domain: domain1 | Server: localhost

GlassFish™ Server Open Source Edition

Baum

Realms

Konfigurationsname: default-config

Realm-Name: SecureRealm

Konfigurationsname: default-config

Realms

JAAS-Konfigurationen

Schlüsselmaterialien

Gruppen

Weitere Optionen

Eigenschaften

Select

Keine Elemente

Realms

SecureRealm

admin-realm

certificate

file

Auditmodule

JACC-Provider

Nachrichtensicherheit

Systemeigenschaften

Threadpools

Transaktionsservice

Virtuelle Server

Webcontainer

server-config

Dateibenutzer

Verwalten Sie Benutzeraccounts für die derzeit ausgewählte Konfiguration.

Konfigurationsname: default-config

Realm-Name: SecureRealm

Dateibenutzer (3)

Neu... Löschen

Select Benutzer-ID

User2

User1

Admin

User und Gruppen

Group / Role mapping

glassfish-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD
  3.1//EN//EN" "http://glassfish.org/dtds/glassfish-web-app_3_1.dtd">
<glassfish-web-app error-url="<error-page><location>/error</location></error-page></error-url>>
  <context-root>/app</context-root>
  <security-role-mapping>
    <role-name>administrator</role-name>
    <principal-name>Admin</principal-name>
    <group-name>administrator</group-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>user</role-name>
    <principal-name>User1</principal-name>
    <group-name>user</group-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>user</role-name>
    <principal-name>User2</principal-name>
  </security-role-mapping>
</glassfish-web-app>
```

Java EE 7 Security – Grundlagen: Sicherheitsmodelle

Man unterscheidet drei unterschiedliche Sicherheitsmodelle:

1. Deklarative Sicherheit

- ▶ Über File-Deskriptoren (web.xml, ejb-jar.xml oder proprietäre Dateien)
- ▶ Über Annotationen innerhalb der Java-Klassen

2. Programmatische Sicherheit

- ▶ Teil des Applikationscodes

3. Messagebasierte Sicherheit (nicht Teil der JEE Spezifikation)

- ▶ Mit digitalen Signaturen und Verschlüsselungen auf Message-Layer (innerhalb einer SOAP-Message [xml encryption, xml signature])

Java EE 7 Security – Grundlagen: Sicherheitsmodelle

1. Deklarative Sicherheit

- Definitionen in **Deployment-Deskriptoren** (Standard-Deskriptoren und / oder proprietäre Deskriptoren) oder
- Definitionen über **Annotationen** (JSR 250 – Security Annotations, javax.annotation.security siehe auch <https://jcp.org/en/jsr/detail?id=250>) innerhalb der Klassen aber ausserhalb des Programmcodes
- Vergabe von Zugriffsrechten **einzig** über Rollen (nicht über Benutzer)
- Trennung von Businesslogik und Sicherheitsaspekten
- Hintergrund: Die Implementierung der Sicherheit ist Aufgabe des JEE Containers (Web / EJB) und nicht des Programmierers

Java EE 7 Security – Grundlagen: Sicherheitsmodelle



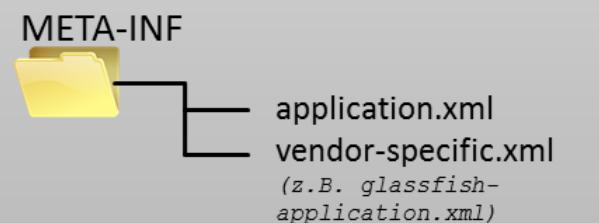
Web Module
WAR – Datei



EJB Module
JAR – Datei



Client Module
JAR – Datei



Application Module
EAR – Datei

- ▶ Einträge in Deployment-Deskriptoren überschreiben Einträge in Annotationen
- ▶ Proprietäre Deployment-Deskriptoren werden teilweise benötigt
- ▶ Proprietäre Deployment-Deskriptoren überschreiben Standard- Deployment-Deskriptoren (in der Regel)

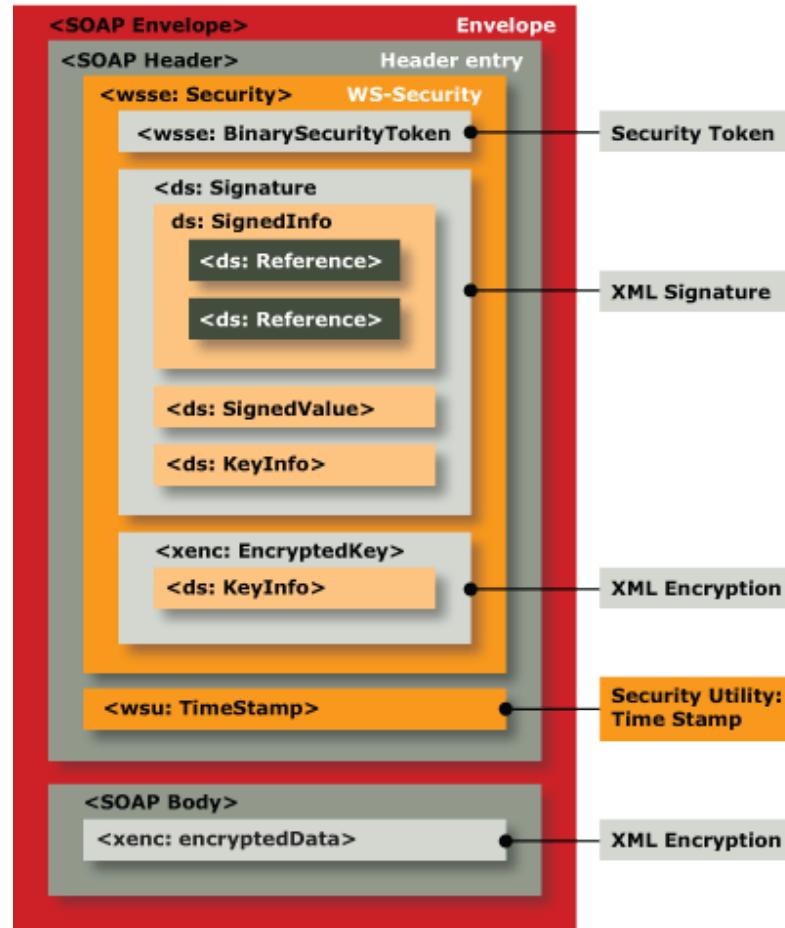
Java EE 7 Security – Grundlagen: Sicherheitsmodelle

2. Programmatische Sicherheit

- Steuerung des Zugriffs **innerhalb des Programmcodes**
- Definition von Rollen in Klassen. Definition deren Berechtigungen innerhalb des Programmcodes
- **Keine Trennung** von Businesslogik (Programmcode) und Sicherheitsaspekten
- Überprüfung von Bedingungen zur Laufzeit
- Hintergrund: Deckt die deklarative Sicherheit die Anforderungen nicht oder nur teilweise ab, kann diese durch die programmatische Sicherheit ergänzt oder ersetzt werden.

Java EE 7 Security – Grundlagen: Sicherheitsmodelle

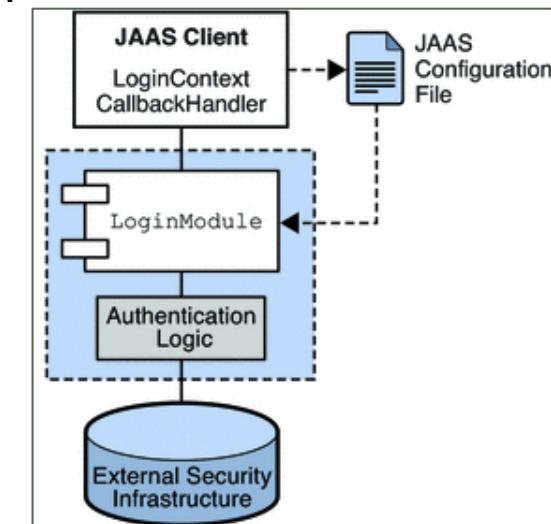
3. Messagebasierte Sicherheit (Web-Service-Security)



Java EE 7 Security – Grundlagen: Security Services

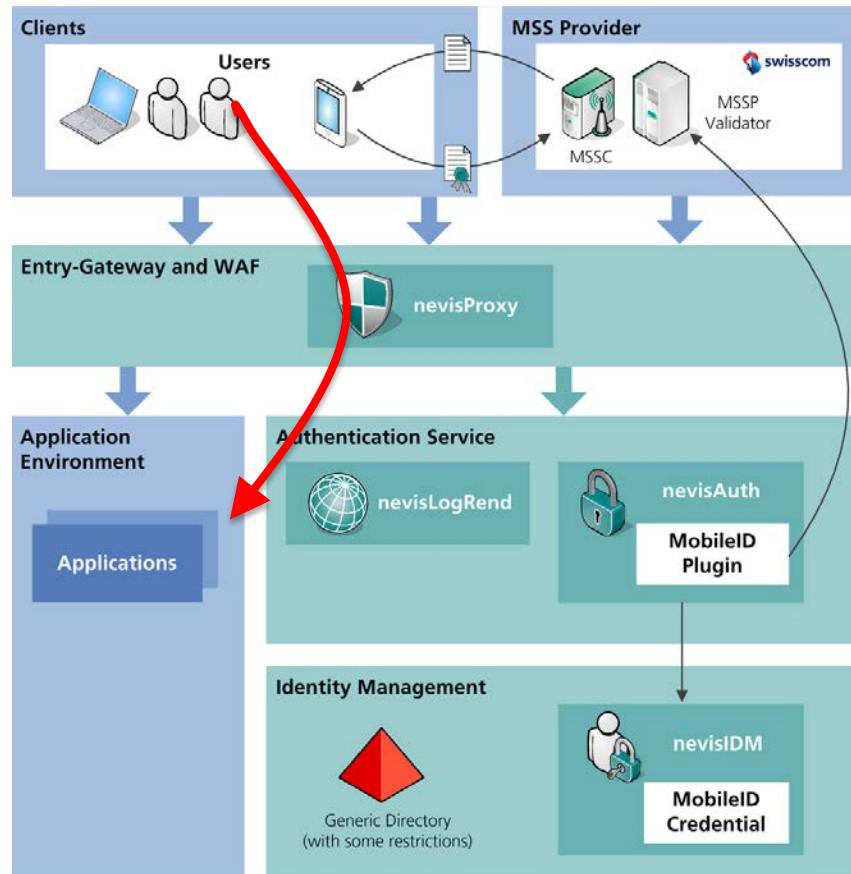
The Java Authentication and Authorization Service (JAAS)

- Authentisierungs-API für pluggable und sequentielle Authentisierung
- Callback-Fähigkeit zwischen Services und Applikationen
- Protokollunabhängige Verarbeitung
- Z.B. um proprietäre Authentisierungen einfach und ohne Anpassungen an Applikationen zu ermöglichen
- Notwendige Interfaces für Authentisierungsmodule:
 - javax.security.auth.*
- Links und Tutorial
 - <https://jcp.org/en/jsr/detail?id=115>
 - <https://docs.oracle.com/javase/7/docs/technotes/guides/security/jaas/tutorials/GeneralAcnOnly.html>
 - <https://www.youtube.com/watch?v=xv2xltxnnBU>



Java EE 7 Security – Grundlagen: Security Services

JAAS-Login-Modul: Hier am Beispiel im Zusammenspiel mit einem Proxy



Java EE 7 Security – Grundlagen: Security Services

Java Authentication SPI for Containers (JASPI, JSR 196)

- Standard Service-Provider Interface (SPI) zur Integration eines Authentisierungsmechanismus in «message processing runtimes»
- Definition von Profilen zur Verwendung des SPI in einem spezifischen Kontext
- Notwendige Interfaces für Authentisierungsmodule:
 - `javax.security.auth.message.module.ClientAuthModule`
 - `javax.security.auth.message.module.ServerAuthModule`
- Tutorials
 - <https://docs.oracle.com/cd/E19226-01/820-7627/girgp/index.html>
 - <https://docs.oracle.com/javaee/7/tutorial/doc/overview007.htm#GIRBE>
 - https://blogs.oracle.com/theaquarium/entry/an_overview_of_jaspic_1

Java EE 7 Security – Grundlagen: Security Services

Java Authorization Contract for Containers (JACC JSR-115)

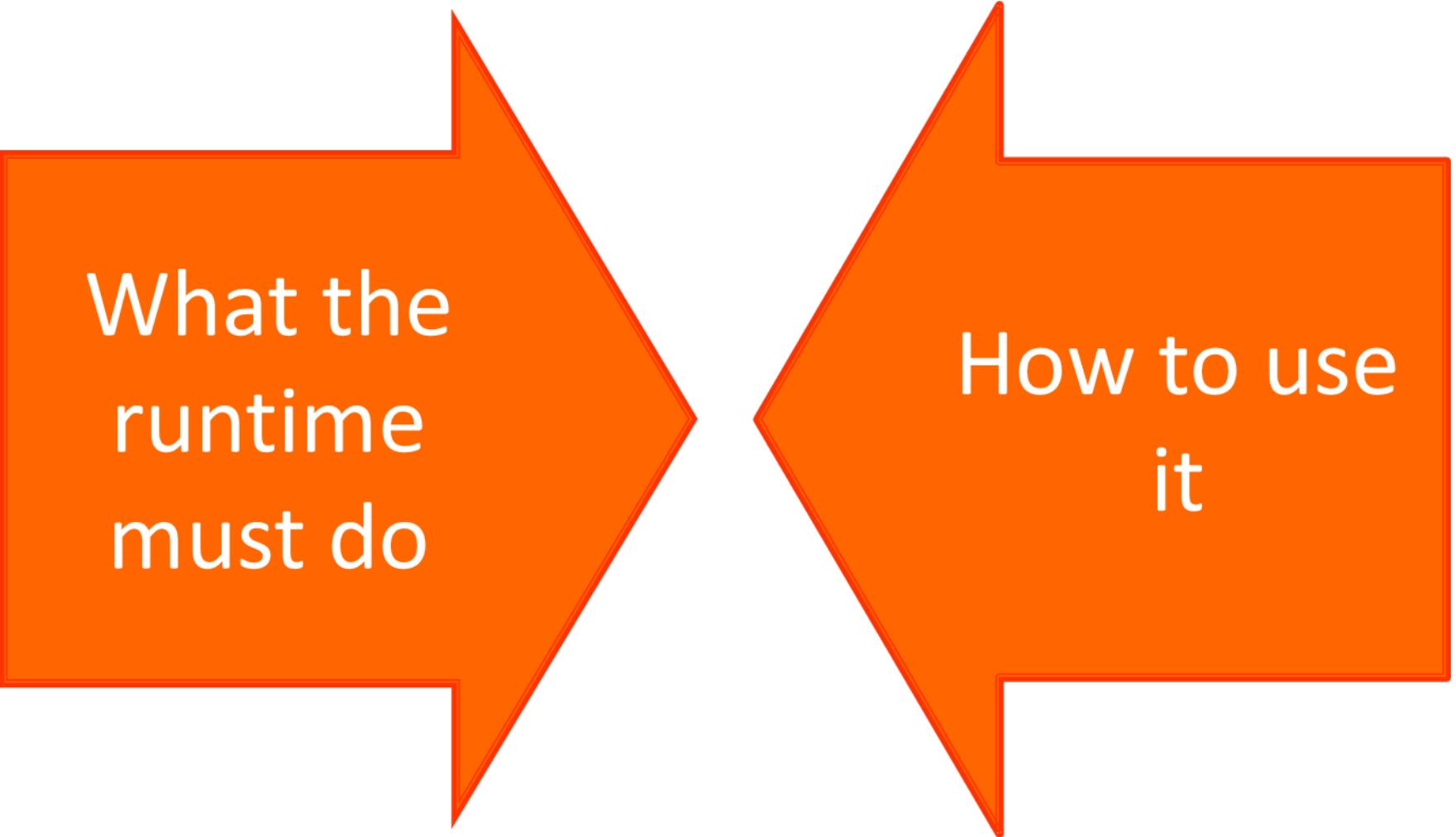
- Definition eines Standardmechanismus für Authorization Provider
- Einführung neuer `java.security.Permission` – Klassen
- Erfüllung neuer JEE-Anforderung an die Autorisierung:
 - Definition von Rollen als Sammlung von Berechtigungen
 - Zuweisung von Berechtigungen zu Principal mittels Rollen
- Notwendige Interfaces :
 - `javax.security.jacc*`
- Tutorials
 - https://docs.oracle.com/javaee/7/api/javax/security/jacc/package-summary.html#package_description

Java EE 7 Security – Grundlagen: Security Services

Java Generic Security Services (Java GSS-API)

- GSSAPI ist API für Anwendungen, die auf Security Devices zugreifen.
- GSSAPI bietet keine Sicherheit in Form eines API. Stattdessen bieten verschiedene Hersteller ihre Sicherheitssoftware in Form von Libraries an.
- Das wichtigste Feature des GSSAPI ist der Austausch von Nachrichten (Tokens), die die Implementierungsdetails vor den höheren Schichten der Anwendung verstecken.
- Links
 - <https://docs.oracle.com/javase/7/docs/technotes/guides/security/jgss/tutorials/BasicClientServer.html>

Java EE 7 Security – Grundlagen



What the
runtime
must do

How to use
it

Java EE 7 Security Übung 0: Erstellen Realm

- Erstellen eines file-based Realm im Glassfish Applikationsserver mit dem Namen «SecureRealm». Dies kann im Glassfish unter «Konfigurationen->server-konfig->Sicherheit->Realms» erstellt werden.
- Der Klassenname muss «`com.sun.enterprise.security.auth.realm.file.FileRealm`» heissen
- JAAS-Kontext muss «fileRealm» heissen.
- Das File kann unter «\${com.sun.aas.instanceRoot}/config/secure-keyfile» abgelegt werden.
- Erstellen von zwei Gruppen «user» und «administrator»
- Erfassen von folgenden Benutzern:
 - user1 [Gruppe *user*, Password *tbd*]
 - user2 [Gruppe *user*, Password *tbd*]
 - Admin [Gruppe *administrator*, Password *tbd*]
- Aktivieren der Option «default principal-to-role mapping» damit der Glassfish das Role-Mapping **automatisch macht**

Sicherheit im Web-Tier

4.1

Sicherheit im Web

Authentisierungsverfahren

Java EE 7 Security: Authentisierungsverfahren

- Die JEE Plattform unterstützt folgende Authentisierungsverfahren
 - **BASIC** (Basic Authentication), Default - Authentisierung
 - **FORM** (Form-Based Authentication)
 - **DIGEST** (Digest Authentication)
 - **CLIENT-CERT** (Client Certificate Authentication)
- Das Authentisierungsverfahren wird im Deploymentdeskriptor (web.xml) innerhalb des «login-config» Elementes deklariert.
- Im Element «auth-method» unterhalb des Elementes «login-config» wird das Verfahren definiert [BASIC | FORM | DIGEST | CLIENT-CERT]

Konfiguration eines Authentisierungsverfahren [Form] im web.xml

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>myRealm</realm-name>
    <form-login-config>
        <form-login-page>/login/loginform.html</form-login-page>
        <form-error-page>/login/loginerror.html</form-error-page>
    </form-login-config>
</login-config>
```

Java EE 7 Security: HTTP-Status Codes

HTTP Status Codes																						
For great REST services the correct usage of the correct HTTP status code in a response is vital.																						
1xx – Informational	2xx – Successful	3xx – Redirection	4xx – Client Error	5xx – Server Error																		
<p>This class of status code indicates a provisional response, consisting only of the Status-Line and optional headers, and is terminated by an empty line</p> <p>100 – Continue 101 – Switching Protocols 102 – Processing</p>	<p>This class of status code indicates that the client's request was successfully received, understood, and accepted.</p> <p>200 – OK 201 – Created 202 – Accepted 203 – Non-Authoritative Information 204 – No Content 205 – Reset Content 206 – Partial Content 207 – Multi-Status</p>	<p>This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request.</p> <p>300 – Multiple Choices 301 – Moved Permanently 302 – Found 303 – See Other 304 – Not Modified 305 – Use Proxy 307 – Temporary Redirect</p>	<p>The 4xx class of status code is intended for cases in which the client seems to have erred.</p> <p>400 – Bad Request 401 – Unauthorised 402 – Payment Required 403 – Forbidden 404 – Not Found 405 – Method Not Allowed 406 – Not Acceptable 407 – Proxy Authentication Required 408 – Request Timeout 409 – Conflict 410 – Gone 411 – Length Required 412 – Precondition Failed 413 – Request Entity Too Large 414 – Request URI Too Long 415 – Unsupported Media Type 416 – Requested Range Not Satisfiable 417 – Expectation Failed 422 – Unprocessable Entity 423 – Locked 424 – Failed Dependency 425 – Unordered Collection 426 – Upgrade Required</p>	<p>Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request.</p> <p>500 – Internal Server Error 501 – Not Implemented 502 – Bad Gateway 503 – Service Unavailable 504 – Gateway Timeout 505 – HTTP Version Not Supported 506 – Variant Also Negotiates 507 – Insufficient Storage 510 – Not Extended</p>																		
<p>Examples of using HTTP Status Codes in REST</p> <p>201 – When doing a POST to create a new resource it is best to return 201 and not 200. 204 – When deleting a resources it is best to return 204, which indicates it succeeded but there is no body to return. 301 – If a 301 is returned the client should update any cached URI's to point to the new URI. 302 – This is often used for temporary redirect's, however 303 and 307 are better choices. 409 – This provides a great way to deal with conflicts caused by multiple updates. 501 – This implies that the feature will be implemented in the future.</p>																						
<p>Special Cases</p> <p>306 – This status code is no longer used. It used to be for switch proxy. 418 – This status code from RFC 2324. However RFC 2324 was submitted as an April Fools' Joke. The message is <i>I am a teapot.</i></p>																						
<table border="1"> <thead> <tr> <th>Key</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Black</td><td>HTTP version 1.0</td></tr> <tr> <td>Blue</td><td>HTTP version 1.1</td></tr> <tr> <td>Aqua</td><td>Extension RFC 2295</td></tr> <tr> <td>Green</td><td>Extension RFC 2518</td></tr> <tr> <td>Yellow</td><td>Extension RFC 2774</td></tr> <tr> <td>Orange</td><td>Extension RFC 2817</td></tr> <tr> <td>Purple</td><td>Extension RFC 3648</td></tr> <tr> <td>Red</td><td>Extension RFC 4918</td></tr> </tbody> </table>					Key	Description	Black	HTTP version 1.0	Blue	HTTP version 1.1	Aqua	Extension RFC 2295	Green	Extension RFC 2518	Yellow	Extension RFC 2774	Orange	Extension RFC 2817	Purple	Extension RFC 3648	Red	Extension RFC 4918
Key	Description																					
Black	HTTP version 1.0																					
Blue	HTTP version 1.1																					
Aqua	Extension RFC 2295																					
Green	Extension RFC 2518																					
Yellow	Extension RFC 2774																					
Orange	Extension RFC 2817																					
Purple	Extension RFC 3648																					
Red	Extension RFC 4918																					

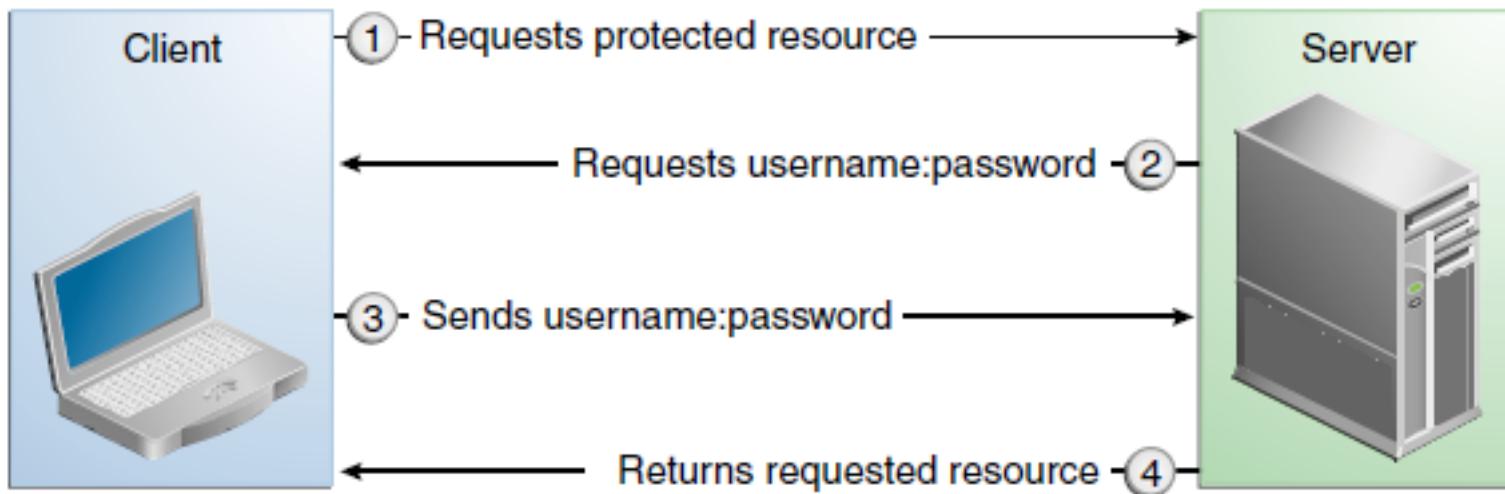
Quelle <http://www.sadev.co.za/content/http-status-codes-cheat-sheet>

Java EE 7 Security: HTTP Methoden

Methode	Beschreibung
GET	Die GET-Methode dient der Anforderung einer HTML-Datei oder einer anderen Quelle. Die Quelle wird durch die URL adressiert. Über GET lassen sich auch Formular-Daten übermitteln. Diese werden in codierter Form der URL angehängt. URL und Formular-Daten sind durch ein Fragezeichen (?) voneinander getrennt.
POST	Die POST-Methode funktioniert ähnlich wie die GET-Methode. POST wird jedoch zur Übermittlung von Formular-Daten an ein Programm oder Skript verwendet. Die Daten werden im Entity-Bereich getrennt durch eine Leerzeile vom Header übertragen.
HEAD	Die HEAD-Methode fordert den Response-Header an, der üblicherweise vom Server nach einem GET-Request übermittelt wird.
PUT	Diese Methode erlaubt das Erstellen oder Ändern von Dateien auf dem Server.
OPTIONS	Die OPTIONS-Methode dient zur Ermittlung von Kommunikationsoptionen durch den HTTP-Client. Es werden jedoch keinerlei Aktionen ausgeführt oder Daten übertragen.
DELETE	Diese Methode führt zur Löschung der Datei, die durch die URL adressiert ist
TRACE	Die TRACE-Methode dient der Verfolgung von HTTP-Requests, die zwischen Client und Server über einen oder mehrere Proxy-Server laufen. Im Header-Feld "Via" des HTTP-Responses sind alle zwischengeschaltete Server protokolliert.
CONNECT	Durch die CONNECT-Methode baut ein Proxy-Server einen Tunnel zum angegebenen Rechner auf und übermittelt darin Daten und Kommandos zwischen Client und Server.

Java EE 7 Security: Authentisierungsverfahren BASIC

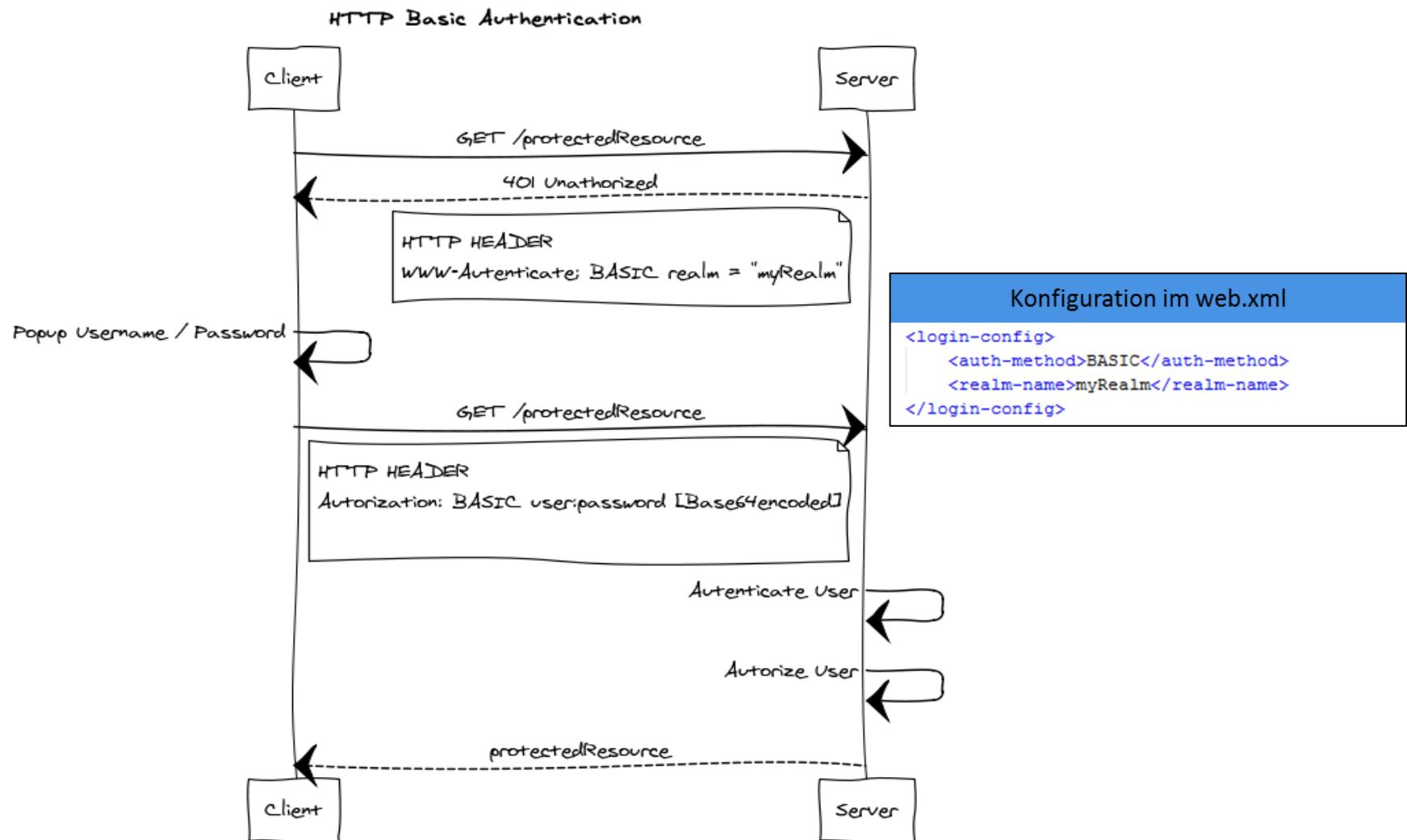
HTTP BASIC Authentication: Übersicht



Quelle: Oracle JEE 7 Tutorial

Java EE 7 Security: Authentisierungsverfahren BASIC

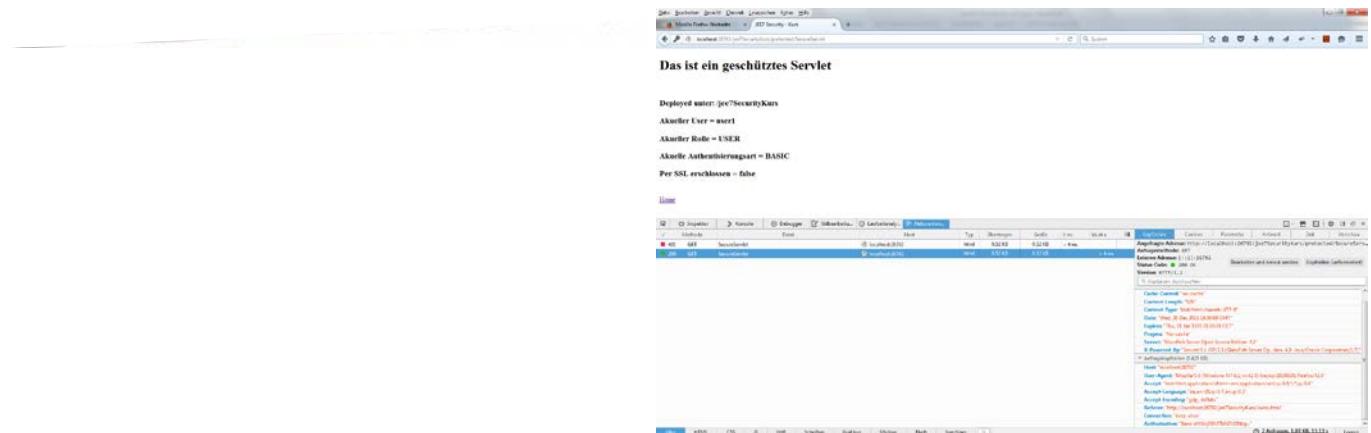
HTTP BASIC Authentication: Detail



Java EE 7 Security: Authentisierungsverfahren BASIC

In der Praxis sieht das dann folgendermassen aus ...

DEMO



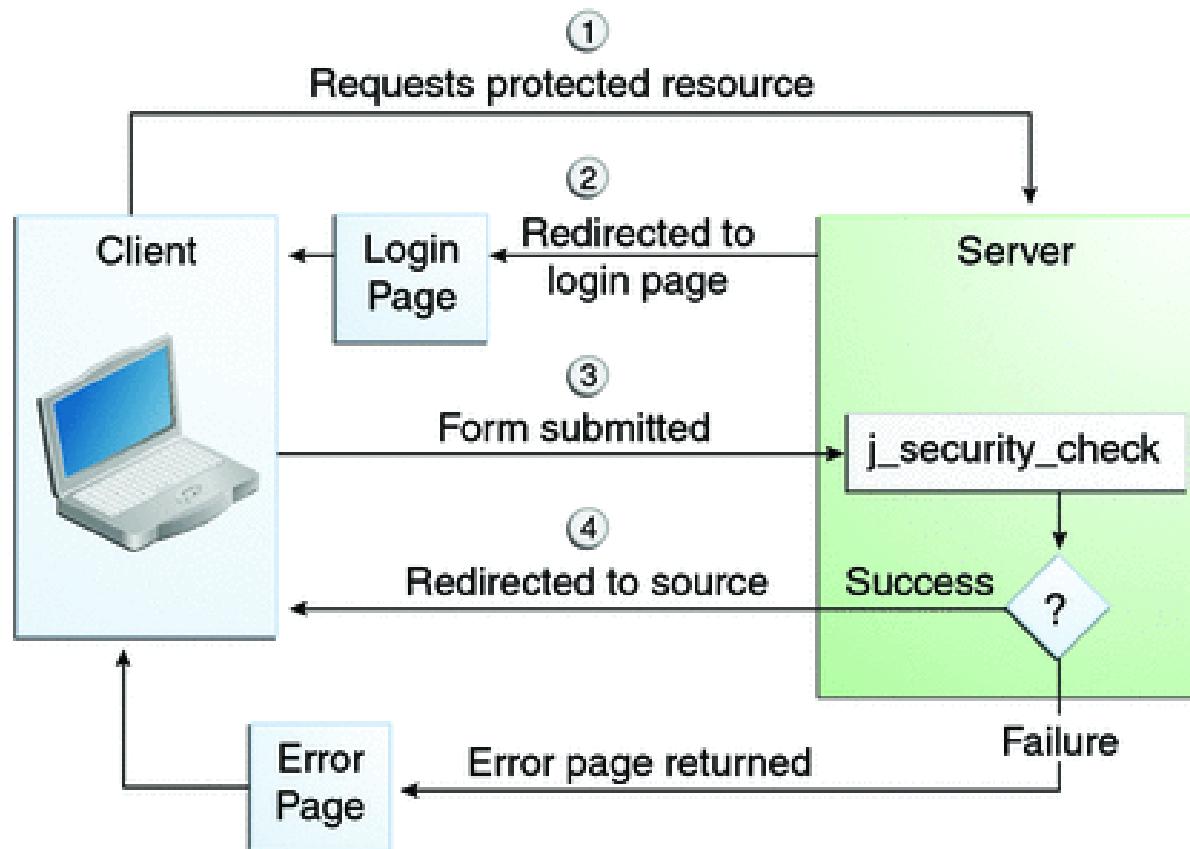
The screenshot shows a web browser window with the following details:

- URL: <http://localhost:8080/BasicAuthProtectedService/basic>
- Page Title: Das ist ein geschütztes Serviet
- Text Content:

```
Deployed under: jee7SecurityKars
Aktueller User = user
Aktueller Rolle = USER
Aktueller Authentisierungsart = BASIC
Per SSL verschlüsseln = false
```
- HTTP Headers:

Name	Value
Cache-Control	no-cache
Content-Type	text/html; charset=UTF-8
Expires	0
Pragma	no-cache
Server	Apache/2.4.18 (Ubuntu) OpenSSL/1.0.2-fips PHP/7.0.24
Set-Cookie	PHPSESSID=63111111111111111111111111111111; expires=Thu, 01-Jan-1970 00:00:00 UTC; path=/; domain=.localhost; HttpOnly
- Network Tab showing a single request to the server.

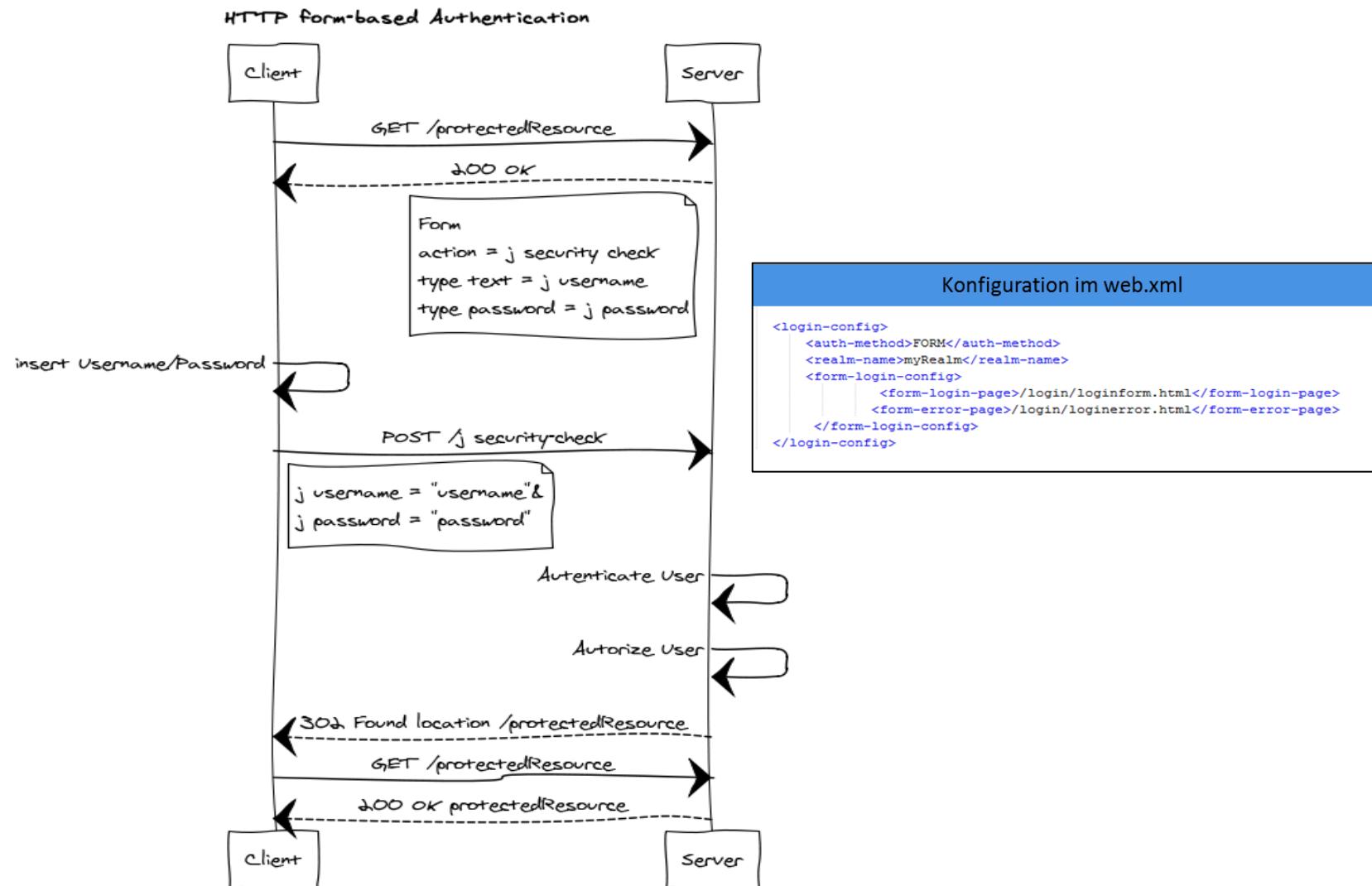
Java EE 7 Security: Authentisierungsverfahren FORM



Quelle: Oracle JEE 7 Tutorial

Java EE 7 Security: Authentisierungsverfahren FORM

HTTP FORM Authentication: Detail



Java EE 7 Security: Authentisierungsverfahren Form

In der Praxis sieht das dann folgendermassen aus ...



>Login Page

localhost:26792/JEE7_S

Bitte einloggen

Username :

Passwort:

Submit Reset

Das ist ein geschütztes Servlet

Deployed unter: JEE7_Security_Kurs_Sample war

Aktueller User = user1

Aktuelle Rolle = USER

Aktuelle Authentisierungsart = FORM

Per SSL verschlossen = false

Header

Content-Language: de-CH

Content-Length: 441

Content-Type: text/html; charset=ISO-8859-1

Date: 13. Mai 2015 15:10:12

Server: Apache/2.4.7 (Ubuntu) OpenSSL/1.0.2g (Ubuntu 1.0.2g-1ubuntu4.14) PHP/5.6.13 (Debian 5.6.13-1) MySQL/5.6.24-0ubuntu0.14.04.1 PHPMyAdmin/4.0.10

X-Powered-By: PHP/5.6.13-1

Set-Cookie: JSESSIONID=40000000000000000000000000000000; Path=/; Secure; HttpOnly

Accept: text/html, application/xhtml+xml, application/xml;q=0.9, */*;q=0.8

Accept-Language: de, en, en;q=0.9, de;q=0.8

Accept-Encoding: gzip, deflate

Cookie: JSESSIONID=40000000000000000000000000000000

Referer: http://localhost:26792/JEE7_Security_Kurs_Sample/protected/protectedServices

User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.130 Safari/537.36

X-Forwarded-For: 127.0.0.1

X-Forwarded-Port: 26792

X-Forwarded-Proto: http

GET /protected/protectedServices

200 OK

Produkt: 2070

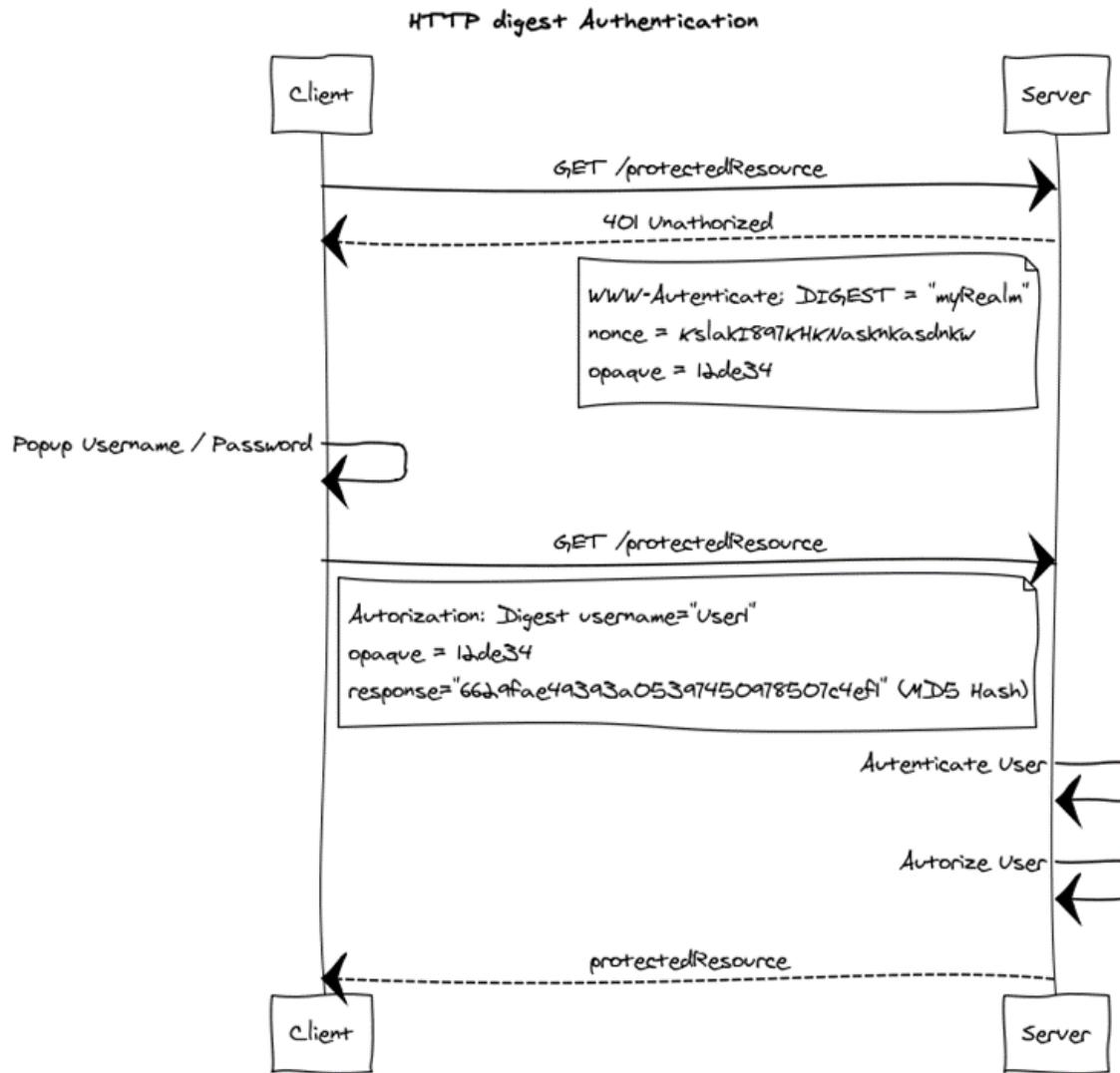
2413 11/12/2015

7678

479ms (inkl. 144ms)

Java EE 7 Security: Authentisierungsverfahren DIGEST

HTTP DIGEST Authentication: Detail



Java EE 7 Security: Authentisierungsverfahren DIGEST

Parameter Authentisierungs-Request Webserver -> Webclient

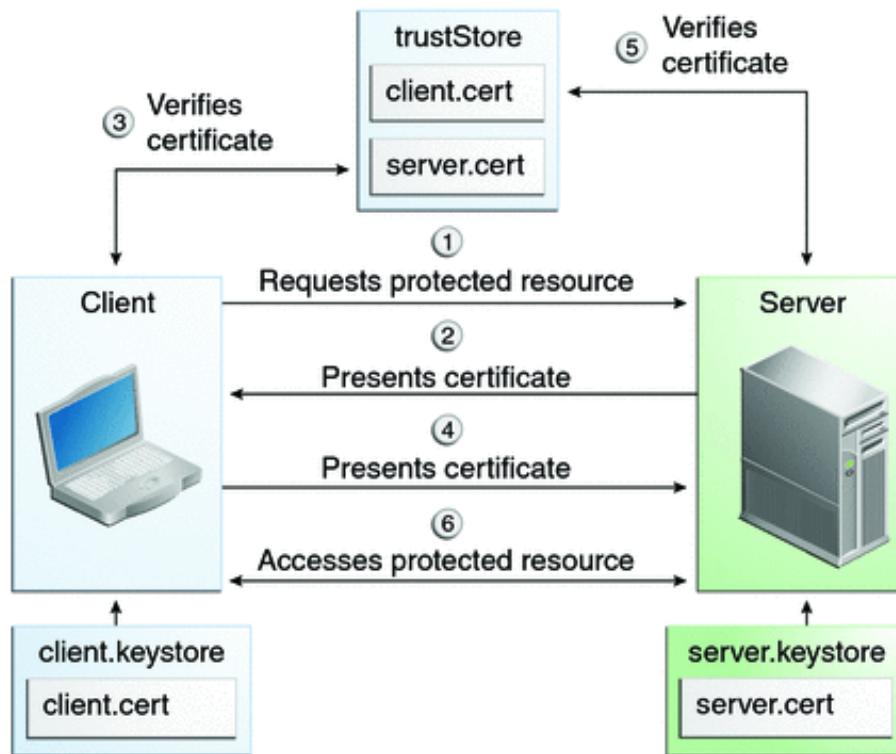
Attribut	Beschreibung
algorithm (optional)	Der zu verwendende Algorithmus. Wenn die Angabe fehlt, wird MD5 angenommen.
domain (optional)	Koma-separierte Liste von URIs
nonce	Ein Zufalls-String, der serverseitig erzeugt wird. Dieser String wird dann clientseitig zusammen mit dem Passwort benutzt, um daraus den Digest zu bestimmen. Wird nur einmal verwendet.
opaque (optional)	Ein Zufalls-String, der serverseitig erzeugt wird. Dieser muss unverändert an den Server zurückgeschickt werden.
qop	Zeigt die Qualität an. (Quality of protection) [auth" oder "auth-int"]
realm	Der Name einer geschützten Security-Domäne.
state	Ein boolesches Attribut. Der Wert true bedeutet, dass der Wert von nonce veraltet ist und der Webclient nochmal ein Request machen muss.

Java EE 7 Security: Authentisierungsverfahren DIGEST

Parameter Authentisierungs-Request Webclient -> Webserver

Attribut	Beschreibung
qop	Quality of protection, welche der Client angewendet hat [auth" oder "auth-int"]
cnonce	Nonce vom Client
nc	Nonce Count (vermeiden mehrfachem senden von Requests)
response	32 hex String mit Username und Passwort (gehashed)
uri	Der URI der Anfrage
opaque	Ein Zufalls-String, der serverseitig erzeugt wird. Dieser muss unverändert an den Server zurückgeschickt werden.
algorithm	Der zu verwendende Algorithmus. Wenn die Angabe fehlt, wird MD5 angenommen.

Java EE 7 Security: Authentisierungsverfahren CLIENT-CERT

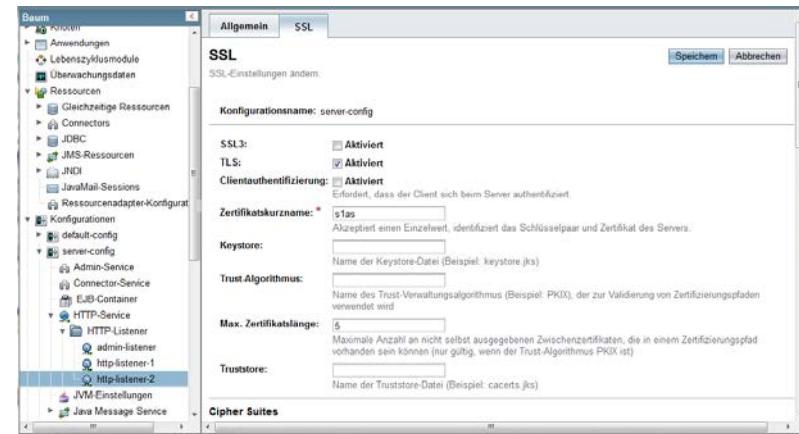


Quelle Oracle JEE 7 Tutorial

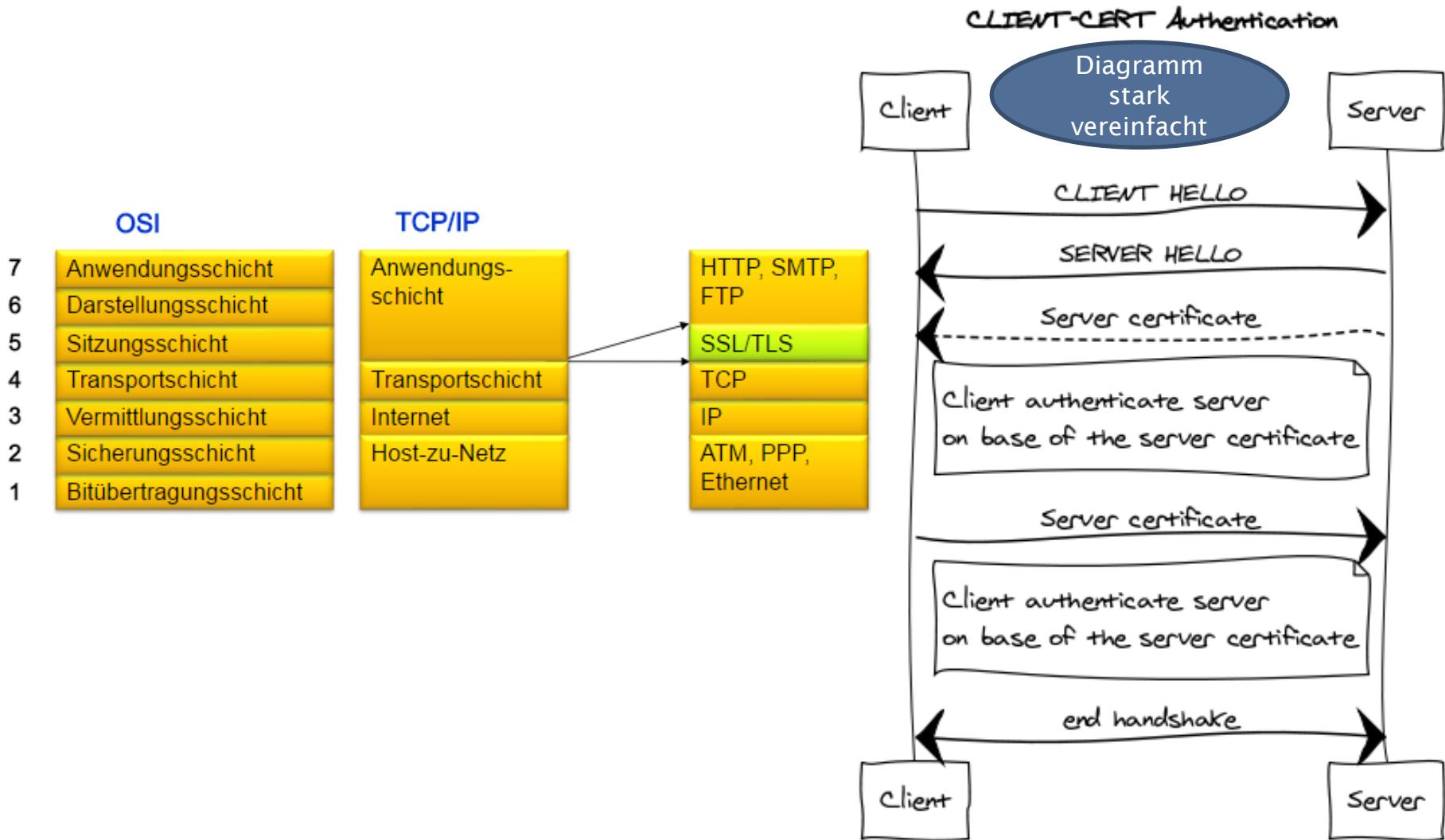
Aktivierung über den Deployment-Deskriptor



Aktivierung über das Glassfish-WebGUI

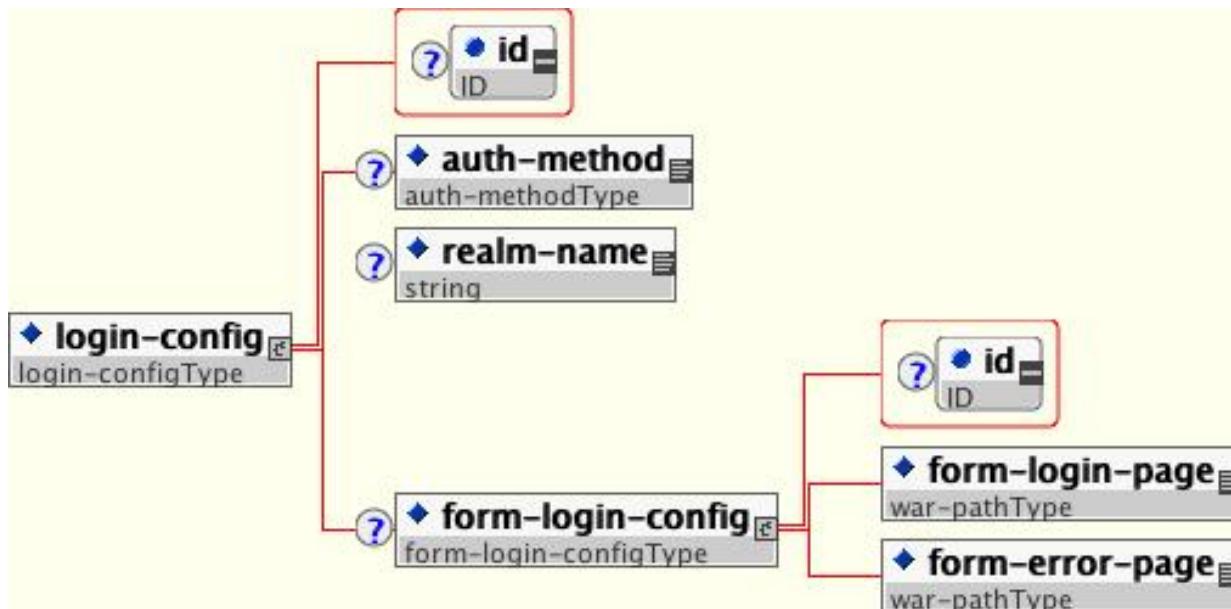


Java EE 7 Security: Authentisierungsverfahren CLIENT-CERT



Java EE 7 Security: Authentisierungsverfahren

Übersicht des Schemas / Elementes (*login-config*) im *web.xml*



Java EE 7 Security: Authentisierungsverfahren

Ein Sicherheits-Vergleich der Authentisierungeverfahren

Methode	Sicherheits-niveau	Aufwand für Implementierung	Server-anforderungen	Kommentare
Standard-Authentisierung	Niedrig	Niedrig	Benutzerverwaltung	Authentisierungsinformationen und Daten werden unverschlüsselt übertragen!
Formularbasierte Authentisierung ohne gesicherte Übertragung	Niedrig	Niedrig bis mittel	Implementierung in der jeweiligen Anwendung	Authentisierungsinformationen und Daten werden unverschlüsselt übertragen!
Digest-Authentisierung	Mittel	Niedrig	Benutzerverwaltung	Daten werden unverschlüsselt übertragen.
Formularbasierte Authentisierung über SSL	Hoch	Mittel bis hoch	SSL-Unterstützung im Server, Implementierung in der jeweiligen Anwendung	Authentisierungsinformationen und Daten werden verschlüsselt übertragen!
Zertifikatbasierte Authentisierung über SSL	Hoch bis sehr hoch	Hoch bis sehr hoch	Installation von Server-Zertifikaten. Zertifikatsverwaltung, Public-Key Infrastruktur.	Wird hauptsächlich für sichere Transaktionen über das Internet verwendet.

Quelle BSI

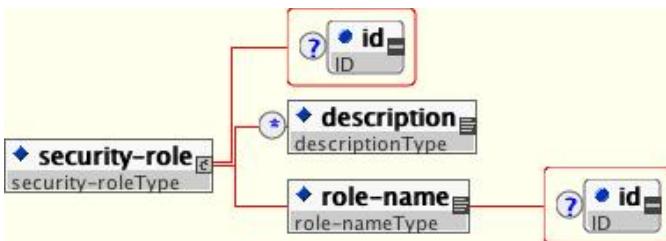
4.2

Sicherheit im Web

Deklarative Sicherheit

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Alle verwendeten Rollen sind zu deklarieren im `web.xml` Deskriptor innerhalb des Elements «`security-role`».



Konfiguration security-role im `web.xml`

```
<security-role>
  <description/>
  <role-name>user</role-name>
</security-role>
<security-role>
  <description/>
  <role-name>administrator</role-name>
</security-role>
```

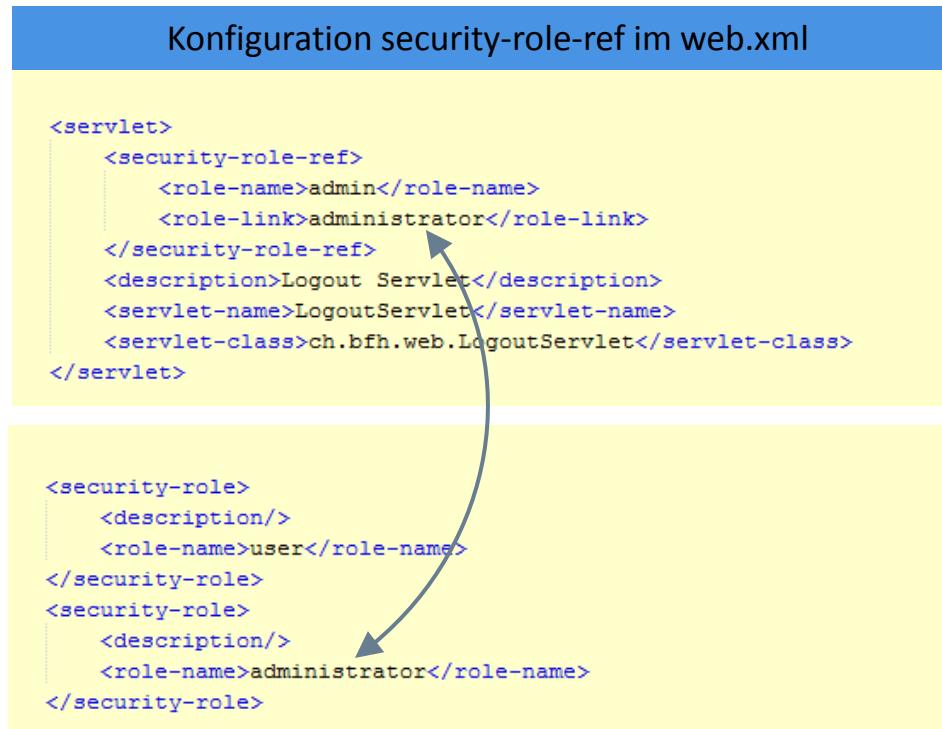
Konfiguration security-mapping im `glassfish-web.xml`

```
<glassfish-web-app>
  <security-role-mapping>
    <!-- 1:1 group mapping, auch default mapping möglich-->
    <role-name>administrator</role-name>
    <group-name>administrator</group-name>
  </security-role-mapping>
  <security-role-mapping>
    <!-- user to Users group mapping, KEIN default mapping möglich-->
    <role-name>user</role-name>
    <group-name>Users</group-name>
  </security-role-mapping>
  <security-role-mapping>
    <!-- user to Users prinzipal mapping, KEIN default mapping möglich-->
    <role-name>user</role-name>
    <principal-name>User007</principal-name>
  </security-role-mapping>
</glassfish-web-app>
```

Und zu «mappen» im proprietären Deskriptor des Application servers

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Im `web.xml` können (optional) im Element `<security-role-ref>` Rollen verlinkt werden



Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Absichern (*security-constraint*) von Ressourcen via Deskriptor (*web.xml*)

Listing: security-constraint im web.xml

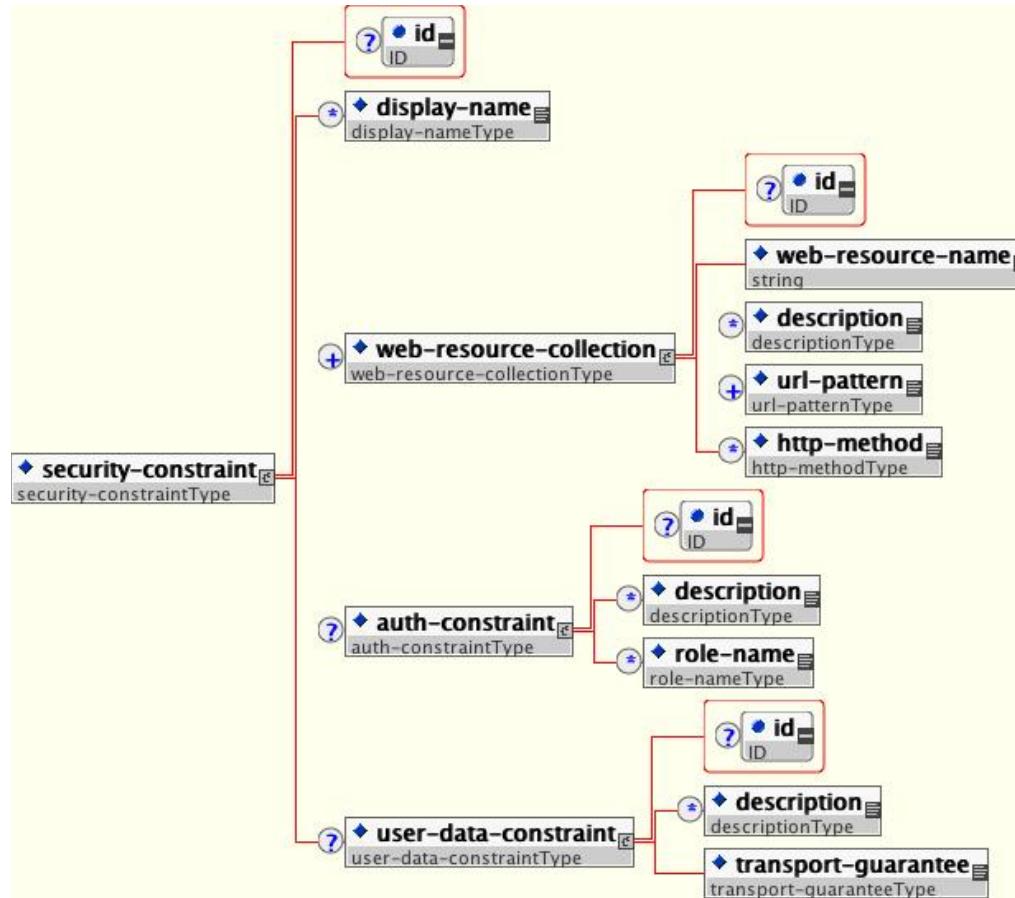
```
<security-constraint>
  <display-name>protected ressources</display-name>
  <web-resource-collection>
    <web-resource-name>Protected</web-resource-name>
    <description>Matches all pages from protected</description>
    <url-pattern>/protected/*</url-pattern>
    <http-method>POST</http-method>
    <http-method>PUT</http-method>
    <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>administrator</role-name>
  </auth-constraint>
  <user-data-constraint>
    <description>SSL not required</description>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

The diagram illustrates the mapping of XML elements in a security-constraint to their respective security descriptors. The XML code is on the left, and five boxes on the right explain the meaning of each element. Arrows point from specific XML tags to their corresponding boxes.

- Name (optional)**: Points to the `<display-name>protected ressources</display-name>` element.
- Geschützte Ressourcen (URL)**: Points to the `<url-pattern>/protected/*</url-pattern>` element.
- HTTP Methoden, welche geschützt werden sollen**: Points to the `<http-method>POST</http-method>`, `<http-method>PUT</http-method>`, and `<http-method>GET</http-method>` elements.
- Rollen, welche für diese Ressource zugelassen sind**: Points to the `<role-name>administrator</role-name>` element.
- Definition der Transport-Sicherheit**: Points to the `<transport-guarantee>NONE</transport-guarantee>` element.

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Übersicht des Schemas (*security-constraint*) im *web.xml*



Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Beschreibung der Sub-Elemente des **<security-constraint>** im **web.xml**

<security-constraint> (Teil 1)

Subelement	Konten	Beschreibung
<web-resource-collection>	<web-resource-name>	Name der Ressource
	<description>	Beschreibung
	<url-pattern>	<p>Definition, welche URL (Resoure) geschützt werden soll</p> <ul style="list-style-type: none">• Keine Wildcards ausser * erlaubt• /path/* bedeutet alle Ressourcen unterhalb des entsprechenden Pfades [path-mapping]• *.htm bedeutet alle htm Ressourcen [extension mapping]• Ein leerer String ("") ist ein spezielles URL Pattern, dass genau auf den Root-Kontext passt. In diesem Fall ist die Path Info "/"• Alle anderen url-patterns (Strings) müssen genau auf die Ressourcen passen

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Beschreibung der Sub-Elemente des `<security-constraint>` im `web.xml`

`<security-constraint>` (Teil 2)

Subelement	Konten	Beschreibung
<code><web-resource-collection></code>	<code><http-method></code>	<p>Definition, welche HTTP Methode in der constraint geschützt werden soll</p> <ul style="list-style-type: none">• Keine Angabe bedeutet, dass alle HTTP-Methoden durch die constraint geschützt sind.• Explizit angegebene HTTP-Methoden bedeuten, dass genau die angegebenen Methoden durch die constraint geschützt sind. Alle anderen Methoden sind ungeschützt
	<code><http-method-omission></code>	<p>Definition, welche HTTP Methode in der constraint vom Schutz ausgeschlossen werden soll</p>

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Beschreibung der Sub-Elemente des `<security-constraint>` im `web.xml`

`<security-constraint>` (Teil 3)

Subelement	Konten	Beschreibung
<code><auth-constraint></code>	<code><description></code>	Beschreibung
	<code><role-name></code>	<p>Name der Rolle, welche Zugriff auf die im Element <code><web-resource-collection></code> definierte Ressource haben soll.</p> <ul style="list-style-type: none">• Ist das <code><auth-constraint></code> Element leer, wird kein Zugriff auf das in der constraint definierte Ressource gewährt• Fehlt das <code><auth-constraint></code> Element ganz, wird keine Authentisierung / Autorisierung durchgeführt• <code><role-name>* </role-name></code> ist eine Definition dass alle Rollen Zugriff haben (welche in <code><security-role></code> definiert wurden)• <code><role-name>** </role-name></code> ist eine Definition dass alle User (unabhängig der Rolle) Zugriff haben• Jede Rolle im <code><auth-constraint></code> Element muss auch im Element <code><security-role></code> definiert sein

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Beschreibung der Sub-Elemente des *<security-constraint>* im *web.xml*

<security-constraint> (Teil 4)

Subelement	Konten	Beschreibung
<user-data-constraint>	<description>	Beschreibung
	<transport-guarantee>	<p>Definition wie resp ob eine Transportverschlüsselung aktiviert werden soll oder nicht</p> <ul style="list-style-type: none">• Werte sind NONE (keine Verschlüsselung), CONFIDENTIAL (Verschlüsselung) oder INTEGRAL (Verschlüsselung)• Es gibt keinen Unterschied zwischen CONFIDENTIAL und INTEGRAL (INTEGRAL wird nicht von allen Servern unterstützt)• CONFIDENTIAL oder INTEGRAL sagt nichts über die Stärke (TLS Version, Chiphers) der Verschlüsselung aus• Fehlt das <user-data-constraint> Element ganz, wird keine Verschlüsselung für das entsprechende constraint forciert.• Das Element <user-data-constraint> darf nicht leer sein.

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Weitere sicherheitsrelevante Elemente im Deskriptor (web.xml)

```
<session-config>
    <session-timeout>30</session-timeout>
</session-config>

<servlet>
    <servlet-name>SecureServlet</servlet-name>
    <security-role-ref>
        <role-name>TheServletRole</role-name>
        <role-link>TheApplicationRole</role-link>
    </security-role-ref>
    <servlet-class>ch.admin.jeeKurs.SecureServlet</servlet-class>
    //use this role for outgoing call's
    <run-as>administrator</run-as>
</servlet>
```

Definiert das Session-Timeout der Applikation

Alias einer Rolle, welche innerhalb des Servlets verwendet werden kann

Das «<run-as>» Element spezifiziert, mit welcher Rolle der Benutzer vom Servlet **ausgehende** Calls ausführt.

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

```
package javax.servlet.annotation;  
@Inherited  
@Documented  
@Target(value=TYPE)  
@Retention(value=RUNTIME)  
  
public @interface ServletSecurity {  
    HttpConstraint value();  
    HttpMethodConstraint[ ] httpMethodConstraints()  
}
```

Annotation	Attribute	Wert / Beschreibung
@ServletSecurity	@HttpConstraint	Constraint, welche auf alle HTTP Methoden wirkt
	@HttpMethodConstraint	Constraint, welche auf eine spezifische HTTP Methoden wirkt

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Absichern (*security-constraint*) von Ressourcen via Annotationen (Servlet)

Listing: Annotation in der Klasse SecureDecServlet.java

```
@WebServlet(name = "SecureDecServlet", urlPatterns = {"/SecureDecServlet/*"})
@ServletSecurity(httpMethodConstraints = {

    @HttpMethodConstraint(
        /*Denying access to all HTTP GET methods
        (all other HTTP methods are allowed) */
        value = "GET",
        /*default authorization semantic, when no roles specified
        by the array rolesAllowed*/
        emptyRoleSemantic = EmptyRoleSemantic.DENY,
        //Specifying that the connection requires no encryption
        transportGuarantee = ServletSecurity.TransportGuarantee.NONE,
        /*Requiring that users must have membership in role "user"
        (for HTTP GET methods)*/
        rolesAllowed = {"user"},

    @HttpMethodConstraint(
        /*Denying access to all HTTP POST methods */
        value = "POST",
        //Specifying that the connection needs encryption
        transportGuarantee = TransportGuarantee.CONFIDENTIAL,
        /*Requiring that users must have membership in role "admin"
        (for HTTP POST methods)*/
        rolesAllowed = "admin")
    )
})
```

Geschützte Ressourcen

HTTP Methoden, welche geschützt werden sollen

Verhalten, wenn keine Rolle definiert wird in «rolesAllowed»

Definition der Transport-Sicherheit

Rollen, welche für diese Ressource zugelassen sind

Zweite Constraint

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

```
package javax.servlet.annotation;  
@Documented  
@Retention(value=RUNTIME)  
  
public @interface HttpConstraint {  
    ServletSecurity.EmptyRoleSemantic value();  
    java.lang.String[] rolesAllowed();  
    ServletSecurity.TransportGuarantee transportGuarantee();  
}
```

Annotation	Attribute	Beschreibung
@HttpConstraint	EmptyRoleSemantic	Definiert das Verhalten, wenn keine Rolle angegeben ist <ul style="list-style-type: none">Default = PERMITWerte EmptyRoleSemantic.PERMIT (alle Zugang) EmptyRoleSemantic.DENY (Keiner Zugang)Optional
	rolesAllowed	Komaseparierte Liste der erlaubten Rollen (Optional)
	transportGuarantee	Definiert ob eine Transportverschlüsselung gefordert ist oder nicht <ul style="list-style-type: none">Default = NONE (wenn Wert nicht angegeben wird)Werte NONE CONFIDENTIALOptional

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

```
package javax.servlet.annotation;  
@Documented  
@Retention(value=RUNTIME)  
  
public @interface HttpMethodConstraint {  
    ServletSecurity.EmptyRoleSemantic value();  
    java.lang.String[] rolesAllowed();  
    ServletSecurity.TransportGuarantee transportGuarantee();  
}
```

Annotation	Attribute	Beschreibung
@HttpMethodConstraint	value	Name der HTTP-Methode (zwingend) [GET POST HEAD usw]
	EmptyRoleSemantic	Definiert das Verhalten, wenn keine Rolle angegeben ist <ul style="list-style-type: none">Default = PERMITWerte EmptyRoleSemantic.PERMIT (alle Zugang) EmptyRoleSemantic.DENY (Keiner Zugang)Optional
	rolesAllowed	Komaseparierte Liste der erlaubten Rollen (Optional)
	transportGuarantee	Definiert ob eine Transportverschlüsselung gefordert ist oder nicht <ul style="list-style-type: none">Default = NONE (wenn Wert nicht angegeben wird)Werte NONE CONFIDENTIALOptional

4.3

Sicherheit im Web

Programmatische Sicherheit

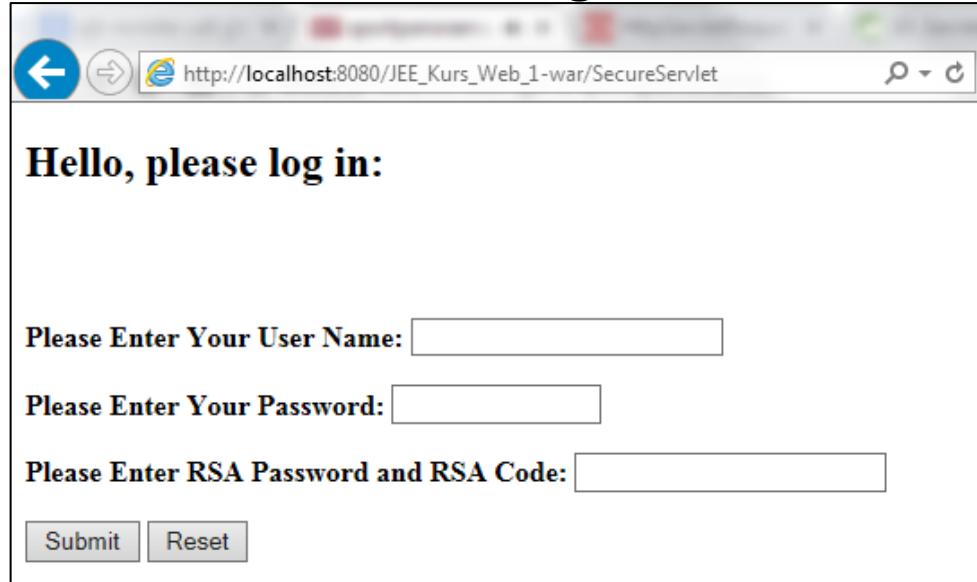
Java EE 7 Security: Reicht deklarative Sicherheit?

Wie kann man sich mit einem “nicht standardisierten” Verfahren authentisieren?



Java EE 7 Security: : Programmatische Sicherheit

Man bräuchte ein Login-Screen wie z.B:



http://localhost:8080/JEE_Kurs_Web_1-war/SecureServlet

Hello, please log in:

Please Enter Your User Name:

Please Enter Your Password:

Please Enter RSA Password and RSA Code:

Submit Reset

und ein Code wie ...

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    String j_username = request.getParameter("j_username");
    String j_password = request.getParameter("j_password");
    String j_rsa = request.getParameter("j_rsa");

    if (!login(j_username, j_password) && login(j_rsa)) {
        throw new SecurityException("User not logged in");
    }
}
```

Java EE 7 Security: Programmatische Sicherheit

Security-Methoden der Klasse HttpServletRequest	Beschreibung
void login(String username, String password)	Methode zum Einloggen des Benutzers (innerhalb des konfigurierten Realm)
void logout()	Methode zum Ausloggen des Benutzers (SecurityContext wird verworfen)
boolean authenticate(HttpServletResponse response)	Forciert ein Login, gemäss konfigurierten "login-config" Element (web.xml), durch den Webserver
boolean isUserInRole(String role)	Prüft ob Benutzer Inhaber einer Rolle ist
String getRemoteUser()	Gibt Username zurück (wenn Benutzer authentisiert ist)
Principal getUserPrincipal()	Gibt das User-Prinzipal zurück (wenn Benutzer authentisiert ist)
String getAuthType()	Gibt die Authentisierungsart zurück (BASIC_AUTH, FORM_AUTH, CERT_AUTH, DIGEST_AUTH oder null)

Java EE 7 Security: Programmatische Sicherheit

Weitere interessante Methoden (HttpServletRequest)	Beschreibung
<code>String getHeader("Authorization")</code>	Gibt in Fall von BASIC Authorization den den Base64 kodierten Authentisierungsstring zurück (BASIC user:password)
<code>String getScheme()</code>	Gibt das verwendete Anfrage-Protokoll zurück, z.B. HTTP, HTTPS oder FTP
<code>isSecure()</code>	Gibt zurück, ob über SSL kommuniziert wurde oder nicht
<code>boolean isRequestedSessionIdValid()</code>	Prüft ob die aktuelle Session noch gültig ist

Java EE 7 Security: Programmatische Sicherheit

Listing: Die Verwendung von HttpServletRequest.isUserInRole

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    if (request.isUserInRole("manager")) {
        response.sendRedirect("/mgr/index.jsp");
    } else {
        response.sendRedirect("/guests/index.jsp");
    }
}
```

Listing: Die Verwendung von HttpServletRequest.login

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    String userName = request.getParameter("user");
    String password = request.getParameter("password");
    try {
        request.login(userName, password);
    } catch (ServletException ex) {
        //Handling Exception
        return;
    }
}
```

Java EE 7 Security Übung 1: Web Security

- Importieren des «JEE7-Security_Kurs_Uebung_1.zip» Projektes in NetBeans
- Deklarieren der benötigten Rollen (Definition des Mapping) im web.xml
- Absichern aller Webressourcen «/protected/*» mit HTTP BASIC Authentication
Verwenden des erstellten Realms «SecureRealm» (Erlaubt sind die Rollen «administrator» und «user», gemäss Konfiguration) mit der Einstellung «default principal-to-role mapping»
- Eine Verschlüsselung ist nicht nötig
- Abhören des Datenverkehrs beim Login auf eine Ressource unter «/protected/*»
- Dabei können Tools wie TCP Monitor, Wireshark, Firebug oder anderer Browser-Entwicklungstool verwendet werden.
- Decodieren des «Authorization» HTTP-Headers (Base64 Encoded)

Beispiel eines HTTP-Header mit den Authorisierungsdaten

```
Accept: text/html, application/xhtml+xml, application/xml;q=0.9, */*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: de, en-US;q=0.7, en;q=0.3
Authorization: Basic VXNlclZGMS1zNDU2Nzg=
Connection: keep-alive
Host: localhost:8080
Referer: http://localhost:8080/JEE_Kurs_Web_1-war/
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:33.0) Gecko/20100101 Firefox/33.0
```

- Zusatzaufgabe – Absichern mit HTTP form based Authentication, dabei können die Formulare «/login/loginform.html» und «/login/loginerror.html» verwendet werden oder eigene Formulare definiert werden
- Ausgeben des User, der Rolle, die Authentisierungsart und die Art der Verschlüsselung

5

Sicherheit im Web-Tier Advanced

Java EE 7 Security: combining constraints

Was passiert, wenn sich constraints überschneiden – sogenannte combining constraints? Es gelten folgende Regeln:

HTTP Methoden

- Ist eine HTTP-Methode (für die selbe URL) einmal geschützt und einmal ungeschützt, bleibt die HTTP-Methode **geschützt**.

Transportsicherheit

- Ein fehlendes `<user-data-constraint>` Element ermöglicht immer einen Zugriff ohne SSL (**egal ob noch eine andere constraint für die entsprechende URL vorhanden ist**)
- Wird für eine URL und eine HTTP-Methode einmal NONE und einmal CONFIDENTIAL verwendet, **wird keine Transportsicherheit (SSL) forciert**.

Java EE 7 Security: combining constraints

Rollen

- Sind in einer `<security-constraint>` keine Rollen für zugelassen (leeres `<auth-constraint>`) Element, können nicht in einem weiteren `<security-constraint>` Rollen erlaubt werden. **Es werden keine Zugriffe gewährt**
- Rollen von unterschiedlichen `<security-constraint>` für die selbe URL werden **kumuliert** (implizite Rollen `«*»` und `«**»` Rollen ebenfalls)
- Eine `<security-constraint>` **ohne ein Element** `<auth-constraint>` ermöglicht immer einen **unauthentisierten** Zugriff auf die entsprechende URL (egal ob noch eine `<security-constraint>` mit einer `<auth-constraint>` für die URL vorhanden ist)
- Eine `<security-constraint>` mit einem **leeren Element** `<auth-constraint>` ermöglicht **keinen** Zugriff (egal ob noch eine `<security-constraint>` mit einer `<auth-constraint>` und Rollen für die URL vorhanden sind)

Java EE 7 Security: combining constraints

Verhaltens-Tipps mit “combining constraints»

- Überschneidungen wenn immer möglich **vermeiden** – da unter Umständen ein nicht deterministisches Verhalten provoziert werden kann (je nach Implementation, siehe auch JEE 7 Tutorial von Oracle)
- Sich gegenseitig ausschliessende constraints vermeiden (z.B. einmal keine Rollen für die HTTP-Methode POST zulassen – einmal die Rolle Admin für die selbe URL für die HTTP-Methode POST zulassen). Dies kann zu nicht erwünschten Effekten führen.
- Wird SSL gefordert mit dem Element <user-data-constraint> (CONFIDENTIAL), ist darauf zu achten, dass bei Überschneidungen der security-constraint (URL-Pattern) immer CONFIDENTIAL gewählt wird, sonst wird die Ressource auch über ungeschützten Transport zugänglich.
- Vorsicht bei SSL – wird einmal auf SSL gewechselt darf nicht mehr auf non-SSL zurückgewechselt werden (Sicherheit !)

Java EE 7 Security Übung 2: combining constraints

- ▶ Analysieren sie die folgenden 4 <security-constraint>
- ▶ Füllen sie die folgende Tabelle aus (SecurityConstraint.xlsx)
- ▶ Markieren sie die Verbindungen, welche durch SSL geschützt sind
- ▶ Präsentieren und diskutieren Sie Ihre Lösungen

<security-constraint>									
<security-constraint>	<url-pattern>	GET	POST	PUT	HEAD	TRACE	DELETE	OPTIONS	
1. constraint	/*								
	/foo/*								
	/bar/*								
2. constraint	/foo/*								
3. constraint	/foo/*								
4. constraint	/bar/*								
Zugriffe (Resultate aller constraints)									
	/*								
	/foo/*								
	/bar/*								
Rollen	USER	ADMIN	OWNER						
Transportverschlüsselung	Ohne	Mit							

Java EE 7 Security Übung 3: combining constraints

<security-constraint> Script 1

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Excluded methods</web-resource-name>
    <url-pattern>/*</url-pattern>
    <url-pattern>/foo/*</url-pattern>
    <url-pattern>/bar/*</url-pattern>
    <http-method-omission>GET</http-method-omission>
    <http-method-omission>POST</http-method-omission>
  </web-resource-collection>
  <auth-constraint/>
</security-constraint>
```

Java EE 7 Security Übung 3: combining constraints

<security-constraint> Script 2

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>User Ressource</web-resource-name>
    <url-pattern>/foo/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>PUT</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>USER</role-name>
  </auth-constraint>
</security-constraint>
```

Java EE 7 Security Übung 2: combining constraints

<security-constraint> Script 3

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>User Ressource</web-resource-name>
    <url-pattern>/foo/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>ADMIN</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transportguarantee>
  </user-data-constraint>
</security-constraint>
```

Java EE 7 Security Übung 3: combining constraints

<security-constraint> Script 4

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Admin Ressource</web-resource-name>
    <url-pattern>/bar/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>ADMIN</role-name>
    <role-name>OWNER</role-name>
  </auth-constraint>
</security-constraint>
```

Java EE 7 Security Übung 3: LoginModul I

Custom-LoginModul in Glassfish 4.0

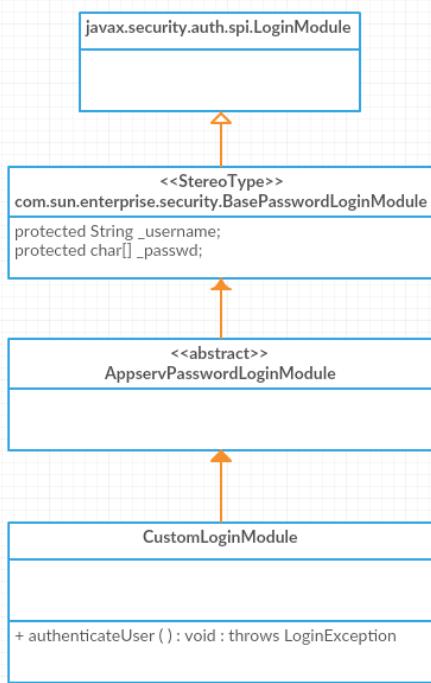
- Ziel ist es, dass sich der User mit einem SecureID-Token authentisieren kann
- Dazu soll keine programmatische Lösung (im Sinne innerhalb der Applikation) erarbeitet werden, sondern ein Custom JAAS – Loginmodul für den Glassfish 4.x
- Um mit einer SecureID einloggen zu können braucht es in der Regel einen Radius Server
- Dieser Radius Server wird «gemockt» mit Hilfe der Klassen
 - `ch.bfh.loginmodule.RadiusAuthentication`
 - `ch.bfh.loginmodule.RadiusMock` (implements `RadiusAuthentication`)
- Die Methode
 - `public boolean authenticate(String username, String password)`
kann verwendet werden um den User zu authentisieren
- Die Mock-Klasse enthält statische User (`user1`)
- Die Mock-Klasse erwartet passwörter gemäss folgendem Pattern
 - `HHMM12345678`, wobei `HHMM` die aktuelle Zeit in Stunden (24h) und Minuten darstellt.

Java EE 7 Security Übung 3: LoginModul I

Anleitung für ein Custom-LoginModul in Glassfish 4.0

Login-Formular (FORM Based authentication)

- Absichern der Applikation mit HTTP FORM Authentication (siehe Übung 2) - ev. ergänzen des Text für das Passwort (HHMM+PW).



JAAS LoginModule und Realm

- Erzeugen des CustomLoginModules (2 Dateien sind nötig)
 - 1) Eine CustomLoginModule - Klasse welche die Klasse com.sun.appserv.security.AppservPasswordLoginModule implementiert
 - 2) Eine CustomRealm Klasse welche die Klasse com.sun.appserv.security.AppservRealm implementiert
- Um kompilieren zu können, müssen im Netbeans-Projekt folgende Libraries eingebunden werden
 - glassfish-ee-api.jar (unter glassfish-4.1\glassfish\modules)
 - security.jar (unter glassfish-4.1\glassfish\modules)

Java EE 7 Security Übung 3: LoginModul II

Klasse CustomLoginModule

- CustomLoginModule welches AppservPasswordLoginModule implementiert
- Überschreiben der Methode authenticateUser in der Klasse CustomLoginModule (die Attributte _username und _passwd können verwendet werden)
- Am Ende (wenn erfolgreich authentisiert) der Methode authenticateUser muss dem User die entsprechende Gruppe (Rolle) zugewiesen werden

```
String[] groups = {"user"};  
commitUserAuthentication(groups);
```

Klasse CustomRealm

- CustomRealm welches AppservRealm implementiert
- Überschreiben der Methode init in der Klasse CustomRealm und setzen des JAAS_CONTEXT («jaas-context»)

```
String propJaasContext=properties.getProperty(JAAS_CONTEXT);  
if (propJaasContext!=null) {  
    setProperty(JAAS_CONTEXT, propJaasContext);  
}
```

Beide Klassen-Body sind bereits im GlassFishLoginModule.zip vorbereitet

Java EE 7 Security Übung 3: LoginModul III

Deployen des Loginmodules (CustomLoginModul)

- ▶ Kopieren des jar-Files (CustomLoginModul) nach
 \glassfish\domains\domainXYZ\lib
- ▶ Editieren der Datei login.conf (
 \glassfishinstallation\glassfish\domains\domainXYZ\config)
- ▶ Registrieren des Realms

```
customloginmoduleRealm {  
    customloginmodule.CustomLoginModule required;  
};
```

Konfigurieren des Glassfish (in der Konsole)

- ▶ Erzeugen eines neuen Realm im Glassfish mit dem Namen «customRealm»
- ▶ Erfassen eines Property mit dem Namen “jaas-context” und dem Wert customLoginModuleRealm, welches so konfiguriert wurde in der Datei login.conf
- ▶ Testen ☺

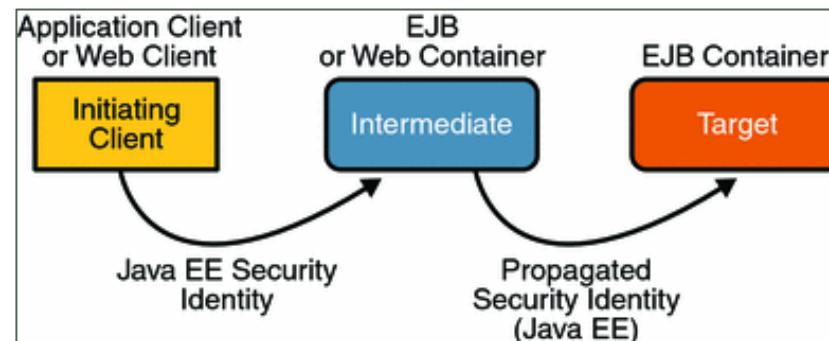
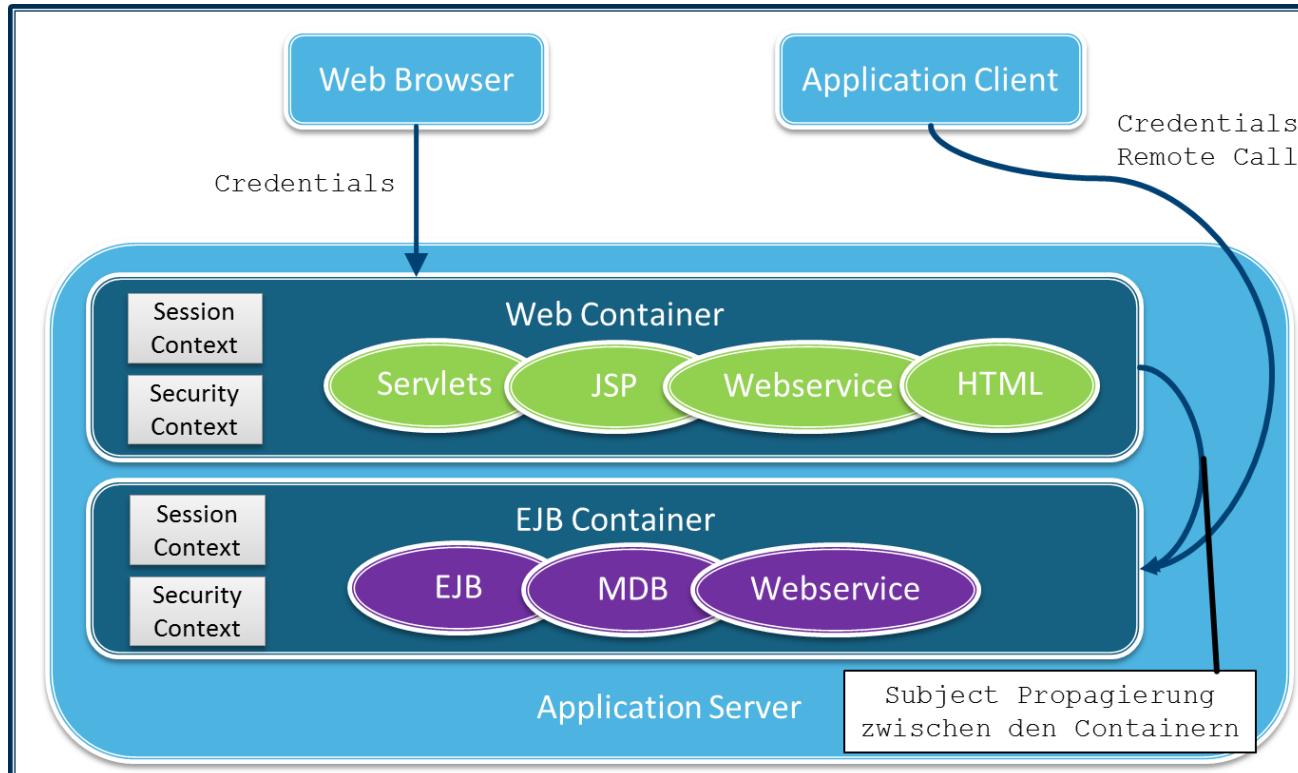
6

Sicherheit im EJB-Tier

Sicherheit im EJB Tier

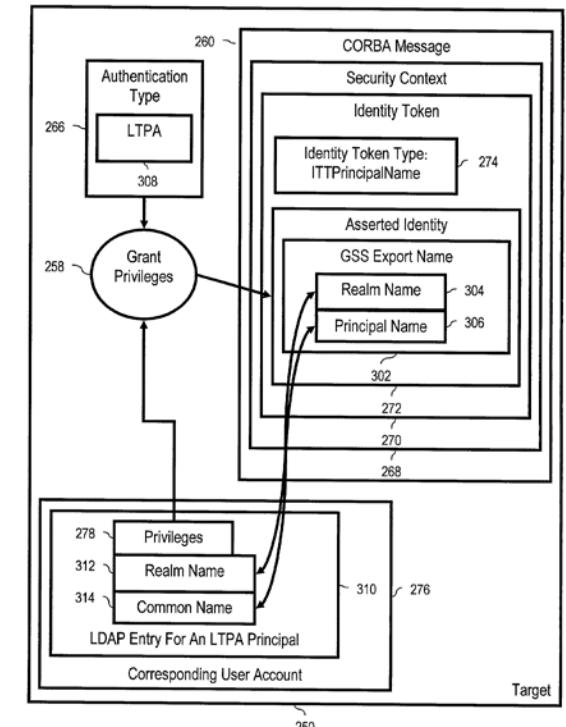
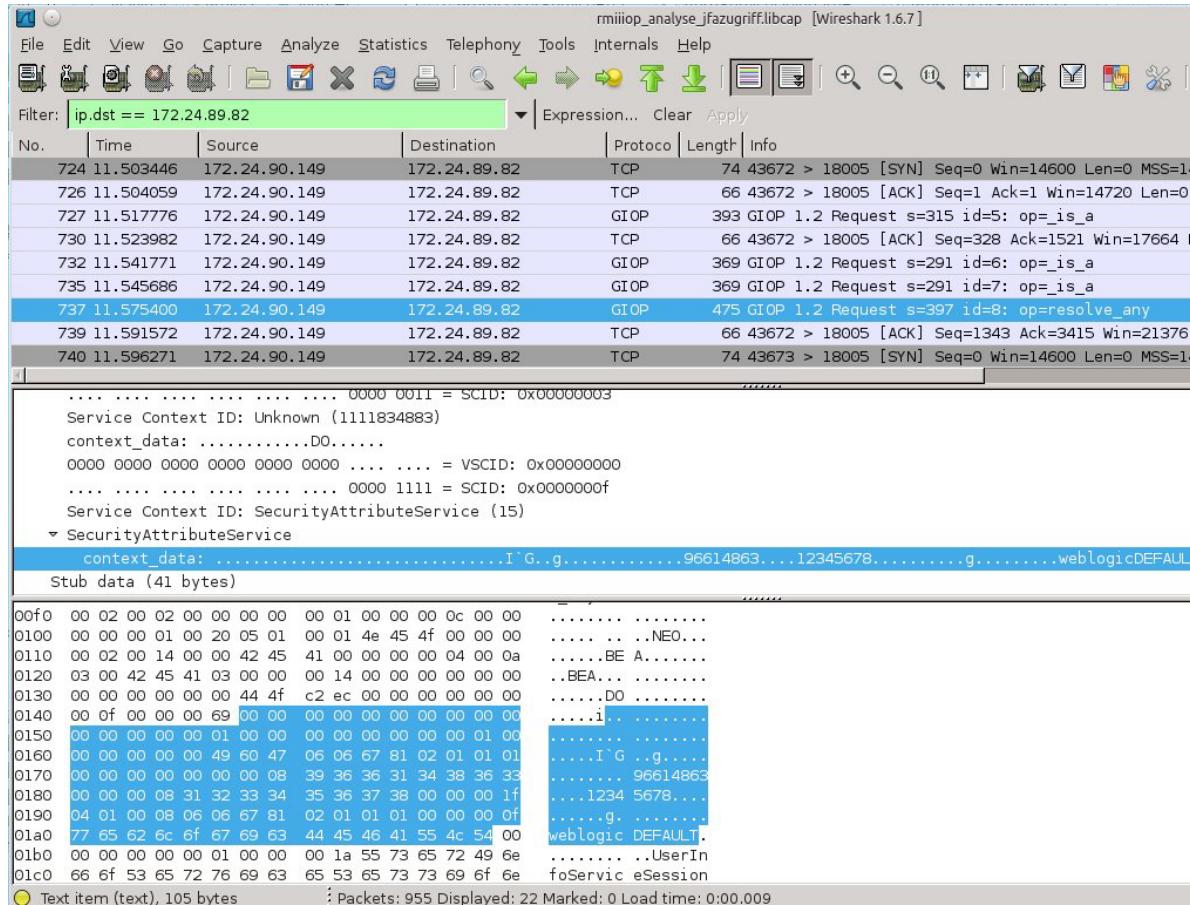
Grundlagen

Java EE 7 Security – EJB-Tier: Grundlagen



Java EE 7 Security – EJB-Tier: Grundlagen

Wireshark – Mitschnitt (Oracle-Weblogic-Server 12.2)



6.1

Sicherheit im EJB Tier

Deklarative Sicherheit



Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Alle verwendeten Rollen sind zu deklarieren im `ejb-jar.xml` Deskriptor

Konfiguration security-role im ejb-jar.xml

```
<assembly-descriptor>
  <security-role>
    <description/>
    <role-name>user</role-name>
  </security-role>
  <security-role>
    <description/>
    <role-name>administrator</role-name>
  </security-role>
```

Konfiguration security-mapping im glassfish-ejb-jar.xml

```
<security-role-mapping>
  <!-- 1:1 group mapping, auch default mapping möglich-->
  <role-name>administrator</role-name>
  <group-name>administrator</group-name>
</security-role-mapping>
<security-role-mapping>
  <!-- user to Users group mapping, KEIN default mapping möglich-->
  <role-name>user</role-name>
  <group-name>Users</group-name>
</security-role-mapping>
<enterprise-beans>
</enterprise-beans>
```

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Absichern (`method-permission`) von EJBs via Deployment-Deskriptor (ejb-jar.xml)

Listing: method-permission im ejb-jar.xml

```
<method-permission>
  <description>Der Administrator darf alle RemoteServiceBean-Methoden aufrufen </description>
  <role-name>Administrator</role-name>
  <method>
    <ejb-name>RemoteServiceBean</ejb-name>
    <method-name>*</method-name>
  </method>
</method-permission>
<method-permission>
  <description>Die Rolle User darf alle nur die readService-Methoden aufrufen </description>
  <role-name>Administrator</role-name>
  <role-name>User</role-name>
  <method>
    <ejb-name>RemoteServiceBean</ejb-name>
    <method-name>readService</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method>
  </method>
</method-permission>
```

Rollen (1-n), welche für diese Ressource zugelassen sind

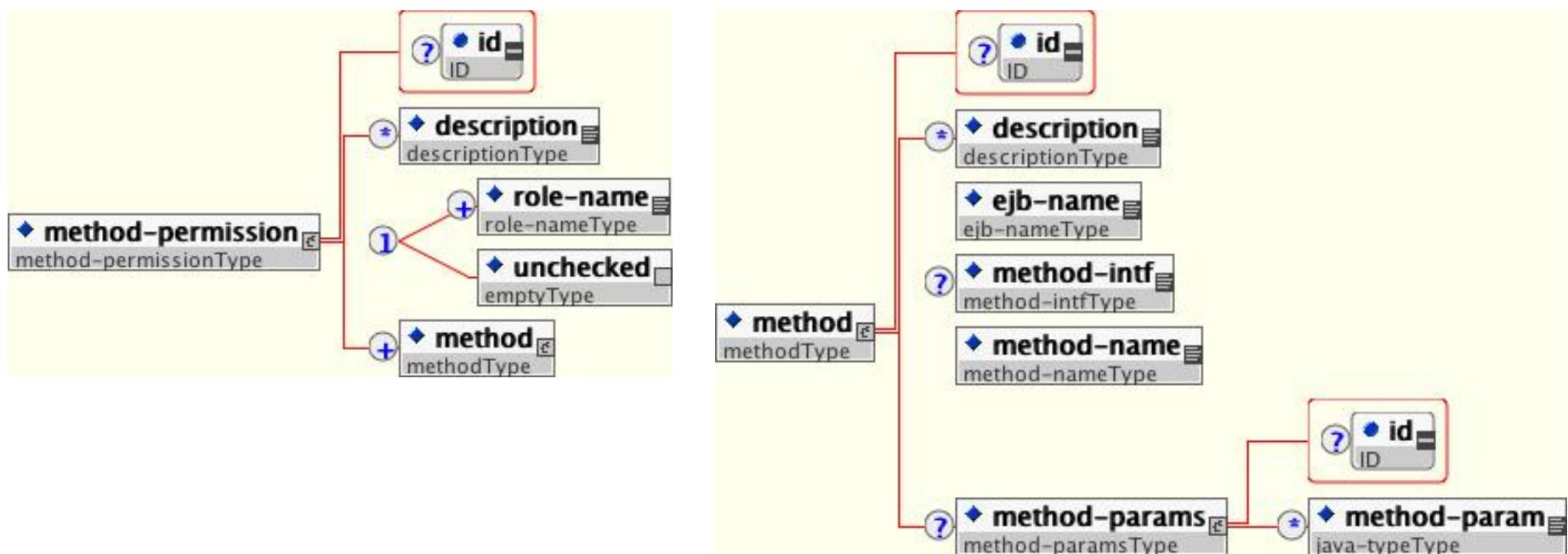
Geschützte Ressourcen (EJB)

Geschützte Methode der Ressource (EJB) [* = alle]

Parameter der geschützten Methode

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Schemas



Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Absichern (method-permission) von EJB via Annotationen

Listing: Annotation in der Klasse SecureSessionBean.java

```
@Stateless
@LocalBean
//nur nötig, wenn innerhalb des EJB die Methode "isCallerInRole" verwendet wird
@DeclareRoles({"user", "administrator"})
public class SecureSessionBean implements ISecureSessionBean {

    @Override
    @RolesAllowed({"user", "administrator"}) //User und Administrator erlaubt
    public String secureUserEcho(String echo) {
        return echo;
    }
    @Override
    @RolesAllowed("administrator") //nur der Administrator
    public String secureAdminEcho(String echo) {
        return echo;
    }
    @Override
    @PermitAll //erlaube allen
    public String secureEcho(String echo) {
        return echo;
    }
    @Override
    @DenyAll //erlaube niemand
    public String internSecureEcho(String echo) {
        return echo;
    }
}
```

Definition der verwendeten Rollen

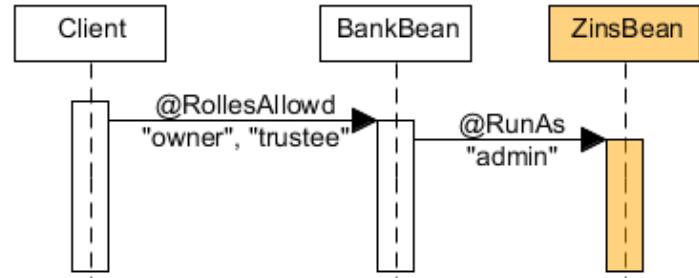
Rollen (1-n), welche für diese Ressource zugelassen sind

Alle Rollen sind für diese Methode zugelassen

Keine Rollen sind für diese Methode zugelassen

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Verwenden der “*run-as*” Annotationen zur «Rollen-Propagierung»



```
@Stateless
@LocalBean
@DeclareRoles({"user", "administrator"})
@RunAs ("Manager")
public class SecureSessionBean implements ISecureSessionBean {

    @Resource
    private SessionContext sctx;

    @EJB
    private RemoteServiceBean rsb;

    @Override
    public String secureEcho(String echo) {
        //Outgoing Call -> propagiert wird die Rolle "Manager"
        rsb.service(echo);
    }
}
```

Die «*run-as*» Annotation wird verwendet, um eine spezifische Rolle bei einem ausgehenden Call zu definieren. Dies ermöglicht es, zwischen Intermediate EJB Container und Target EJB Container unterschiedliche Rollen zu verwenden. Könnte z.B. auch in einem Test (wo der User nicht alle Rollen besitzt) angewendet werden.

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Security-Annotation für Web- und EJB-Container

Annotation	Beschreibung
@ServletSecurity	Das @ServletSecurity kann @HttpMethodConstraint und @HttpConstraint Elemente enthalten für die Absicherung von Servlets.
@DeclareRoles	DeclareRoles definiert analog dem Element <security-role> die verwendeten Rollen.
@RunAs	Definiert die Rolle für ausgehende Aufrufe
@PermitAll	Erlaubt allen Rollen auf die Methode(n) zuzugreifen
@DenyAll	Erlaubt keiner Rolle auf die Methode zuzugreifen
@RolesAllowed	Definiert welche Rollen Zugriff auf die Methode(n) haben

Java EE 7 Security: Deklarative Sicherheit (Deskriptor)

Security-Annotation für Web- und EJB-Container

Annotation	Targets Level	Target Kind
@ServletSecurity	Class	Servlet
@DeclareRoles	Class	EJB, Servlet
@RunAs	Class	EJB, Servlet
@PermitAll	Class, Method	EJB
@DenyAll	Method	EJB
@RolesAllowed	Class, Method	EJB

Sicherheit im EJB Tier

Programmatische Sicherheit

Java EE 7 Security: Programmatische Sicherheit

Listing: Die Verwendung von javax.ejb.EJBContext

```
@Resource
private SessionContext sctx;

@Override
public String secureEcho(String echo) {

    String name = sctx.getCallerPrincipal().getName();
    if (sctx.isCallerInRole("user")){
        //Business für die Rolle "user"
        requestUserData(name);
    } else if (sctx.isCallerInRole("administrator")){
        //Business für die Rolle "administrator"
        requestAdminData(name);
    } else {
        //Business für alle anderen Rolle .
        requestAnonymousData(name);
    }
    return echo;
}
```

Auslesen des aktuellen Users aus dem Prinzipal

Prüfen, ob ein User in einer bestimmten Rolle ist

Ausliefern von Daten anhand der Rolle, welche der Benutzer besitzt

Java EE 7 Security: Programmatische Sicherheit

Security-Methoden der Klasse EJBContext, resp. SessionContext	Beschreibung
public Principal getCallerPrincipal()	Gibt das aktuelle Prinzipal des Aufrufers zurück
String java.security.Principal.getName();	Gibt den Namen des Prinzipals zurück
public boolean isCallerInRole (String roleName)	Prüft zur Laufzeit, ob der aktuelle User die Rolle «roleName» besitzt

Java EE 7 Security: Vergleich

Web Tier	EJB Tier		deklarative Sicherheit	programmatische Sicherheit
Web Resources	Bean Methods	Zugriffskontrolle	Deployment Descriptor	Programm
web.xml	ejb-jar.xml	Deklaration	Container	Programm
Web Container	EJB Container	Umsetzung		
Servlet / JSP	EJB Bean	Codierung	"all or nothing"	Logikbasiert

Java EE 7 Security Übung 4: EJB Security

- Importieren des «JEE7-Security_Kurs_Uebung_4.zip» Projektes in NetBeans
- Absichern des SecureEJB, dabei soll folgendes gelten:
 - echo() dürfen alle aufrufen
 - readInternalData() dürfen alle Benutzer der Gruppe «user» und «administrator» aufrufen
 - readConfidentialData() dürfen nur Benutzer der Gruppe «administrator» aufrufen
- Zeigen Sie 2 Lösungsmöglichkeiten (deklarativ, programmatisch)
- Loggen Sie den Zugriff. Im Log soll enthalten sein
 - User (Username)
 - Methode mit Klassename (ohne Parameter)
- Die Lösung befindet sich «JEE7_Security_Kurs_Uebung_3_Loesung.zip»

7

Sicherheit Webservices (im EJB Container)



Java EE 7 Security : EJB Security - Webservices

- ▶ **EJB-Container (Webservice als EJB implementiert)**
- ▶ Bei annotierten Webservices im EJB-Tier (stateless Sessionbeans) können Annotationen wie `@RolesAllowed`, `@PermitAll` usw. verwendet werden.
- ▶ Jedoch ist die Definition des Loginverfahrens (Verfahren, Realm usw) proprietär gelöst.

Listing: Absichern eines Webservice (JBoss proprietär per Annotation)

```
@Stateless
@WebService(serviceName = "SecureWebService")
@WebContext(contextRoot="/SecureService",
            urlPattern="/Service",
            authMethod="BASIC",
            secureWSDLAccess = false)

@SecurityDomain(value = "jboss-sec-domain")
public class SecureWebService {
```

Listing: Absichern eines Webservice (Glassfish proprietär, glassfish-ejb-jar.xml)

```
<enterprise-beans>
  <ejb>
    <ejb-name>SecureWebService</ejb-name>
    <webservice-endpoint>
      <port-component-name>Service</port-component-name>
      <login-config>
        <auth-method>BASIC</auth-method>
        <realm>SecureRealm</realm>
      </login-config>
    </webservice-endpoint>
  </ejb>
</enterprise-beans>
```

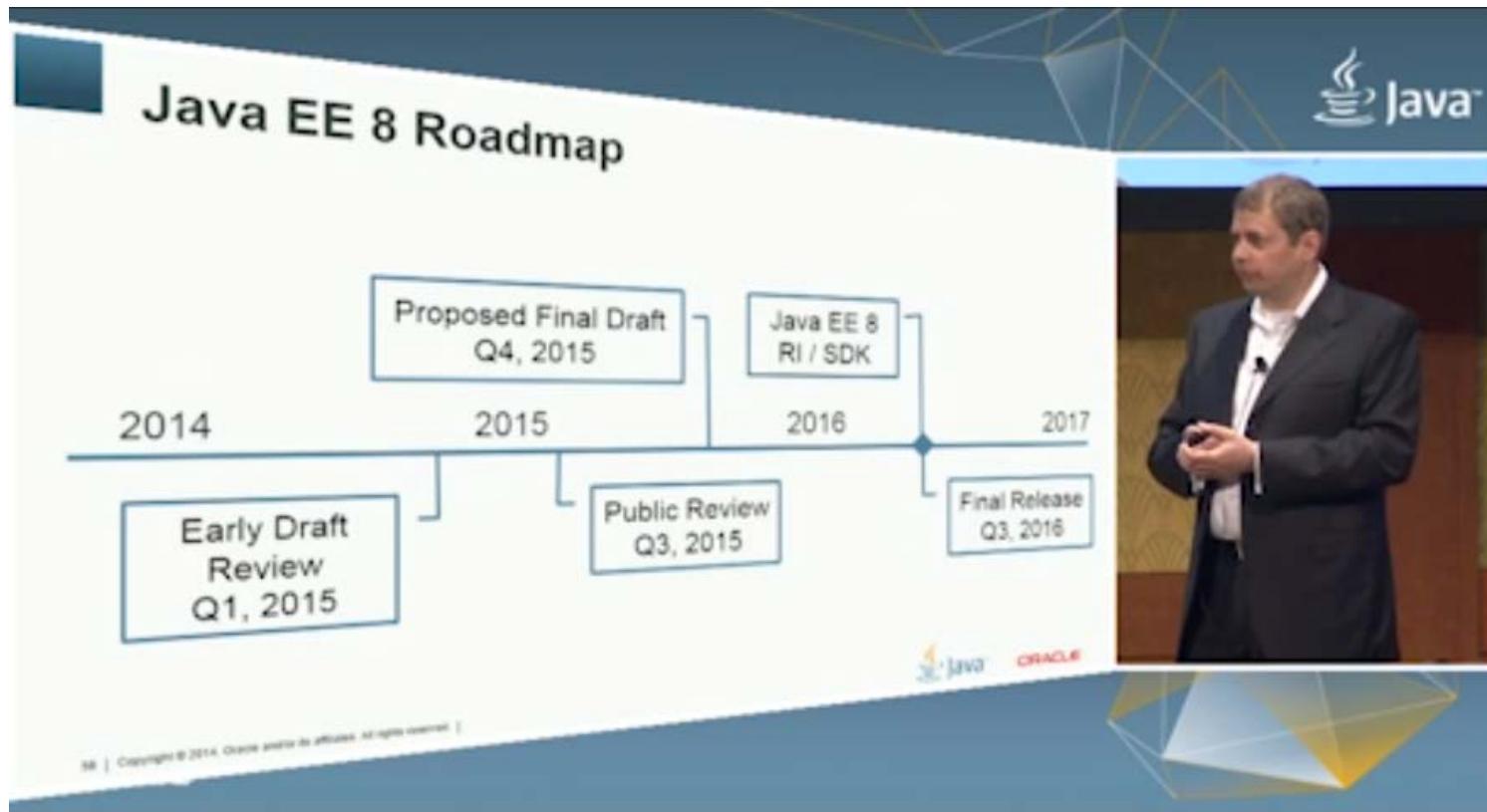
Web-Container (Webservice als Servlet)

Webservices im Web-Tier können wie «normale» Webressourcen im Deployment-Deskriptor (web.xml) abgesichert werden.

8

Ausblick JEE8 (Sicherheit)

Ausblick JEE8 (Sicherheit)



Quelle Oracle

... we are therefore publicly announcing that we are now changing our target time frame for the completion of this **work to the first half of 2017**.

Ausblick JEE8 (Sicherheit)

Neuer JSR (Java Specification Request) für Java Security: JSR 375 - Java EE Security 1.0

- ▶ Das Java EE Security 1.0 (JSR 375) API stellt Einbindung eines vereinfachten Sicherheits-API dar, das Java-EE-Anwendungen die Verwaltung ihrer eigenen Sicherheitsparameter in einzigartiger und doch portabler Art und Weise ermöglicht.
- ▶ Dieser JSR wird ebenfalls für Cloud-basiert Java-EE-Anwendungsbereitstellungen nützlich sein, die auf der Suche nach einer Standardmethode für die Definition von Sicherheitsaspekten sind.
- ▶ Das JSR 375 Proposal (<https://www.jcp.org/en/jsr/detail?id=375>) gibt eine grobe Übersicht, über die Inhalte des neuen API.

Ausblick JEE8 (Sicherheit) JSR 375

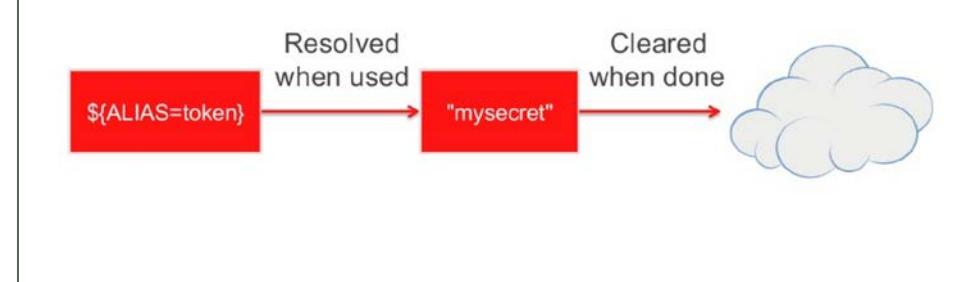
Password aliasing

- ▶ Standardisierter Syntax für das “password aliasing”
 - ▶ Ermöglicht das Weglassen von Klartext-Passwörter im Code

```
@DataSourceDefinition(  
    name="java:app/jdbc/test",  
    user="root",  
    password="${ALIAS=password}",...)
```

- For deployment descriptors

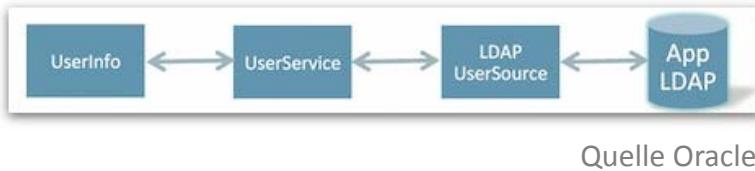
```
<data-source>  
  <name>java:app/env/testDS</name>  
  <user>APP</user>  
  <password>${ALIAS=password}</password>
```



Ausblick JEE8 (Sicherheit) JSR 375

User Management

- ▶ Erlaubt Applikationen ihre eigenen User und Gruppen zu administrieren
 - ▶ Ohne die Applikations-Server-Konfiguration zu verändern
- ▶ User werden in einem applikations-spezifischen Repository (z.B. LDAP) gespeichert
- ▶ Der User-Service [UserService] erlaubt
 - ▶ Erzeugen/Löschen von User und Gruppen, ändern von User / Gruppen, auslesen von Namen usw
- ▶ UserInfo enthält Infos wie Name, Credentials, Rolle(n), weitere Attribute



```
@LdapUserSourceDefinition(  
    name="java:app/ldapUserSource",  
    ldapUrl="ldap://someURL",  
    ldapUser="ElDap",  
    ldapPassword="${ALIAS=LdapPW}",  
    ...  
)  
public class MyAuthenticator {  
    @Resource(lookup="java:app/ldapUserSource")  
    private UserService userService;  
    private boolean isAccountEnabled(String username) {  
        return userService.loadUserByUsername(username).isEnabled();  
    }  
    ...  
}
```

Ausblick JEE8 (Sicherheit) JSR 375

Standardisiertes User-Role-Mapping

- ▶ Role-Mappings können in einem applications-spezifischen Repository gespeichert werden
- ▶ Applikationen können Rollen und Gruppen zu User mappen anhand anwendungsspezifischen Use-Cases
- ▶ Alles ohne die Hilfe des Applikations-Servers



▪ Support in Deployment Descriptors, e.g. web.xml

```
<security-role-map>
  <!-- Role name as set/returned by Authentication Module -->
  <group>MANAGER</group>

  <!-- Role name for mapping -->
  <role-name>EDIT_ACCOUNTS</role-name>
</security-role-map>

<!-- One-to-one group to role mapping -->
<security-role-map groupToRoleMapping="false" />
```

```
@EmbeddedRoleMapper(
  users={
    @RoleMap(user="foo",roles="admin"),
    @RoleMap(group="admin",roles={"admin","manager"})
  }
)
public class MyServlet {
```

Quelle Oracle

Ausblick JEE8 (Sicherheit) JSR 375

Authorization Interceptors

- Die Authentisierung kann neu nicht nur anhand von Rollen definiert werden, sondern es kann eine rule-based security annotiert werden

https://blogs.oracle.com/theaquarium/entry/javaone_replay_java_ee_8

- EL Authorization rules centrally managed in a repository

```
@LdapAuthorizationRules (
    name="java:app/accountAuthRules",
    ldapUrl="ldap://blah",
    ldapUser="ElDap",
    ldapPassword="mysecret"
)
public class MyBean {
    @EvaluateSecured(
        ruleSourceName="java:app/accountAuthRules", rule="transferFunds")
    void transferFunds() {...};
    ...
}
```

```
@IsAuthorized("hasRoles('Manager') && schedule.officeHours")
public void transferFunds() ;

@IsAuthorized(
    "hasRoles('Manager') && hasAttribute('directReports', employeeId) ")
public double getSalary(long employeeId) ;

@IsAuthorized(ruleSourceName="java:app/payrollAuthRules")
public void displayReport() ;
```

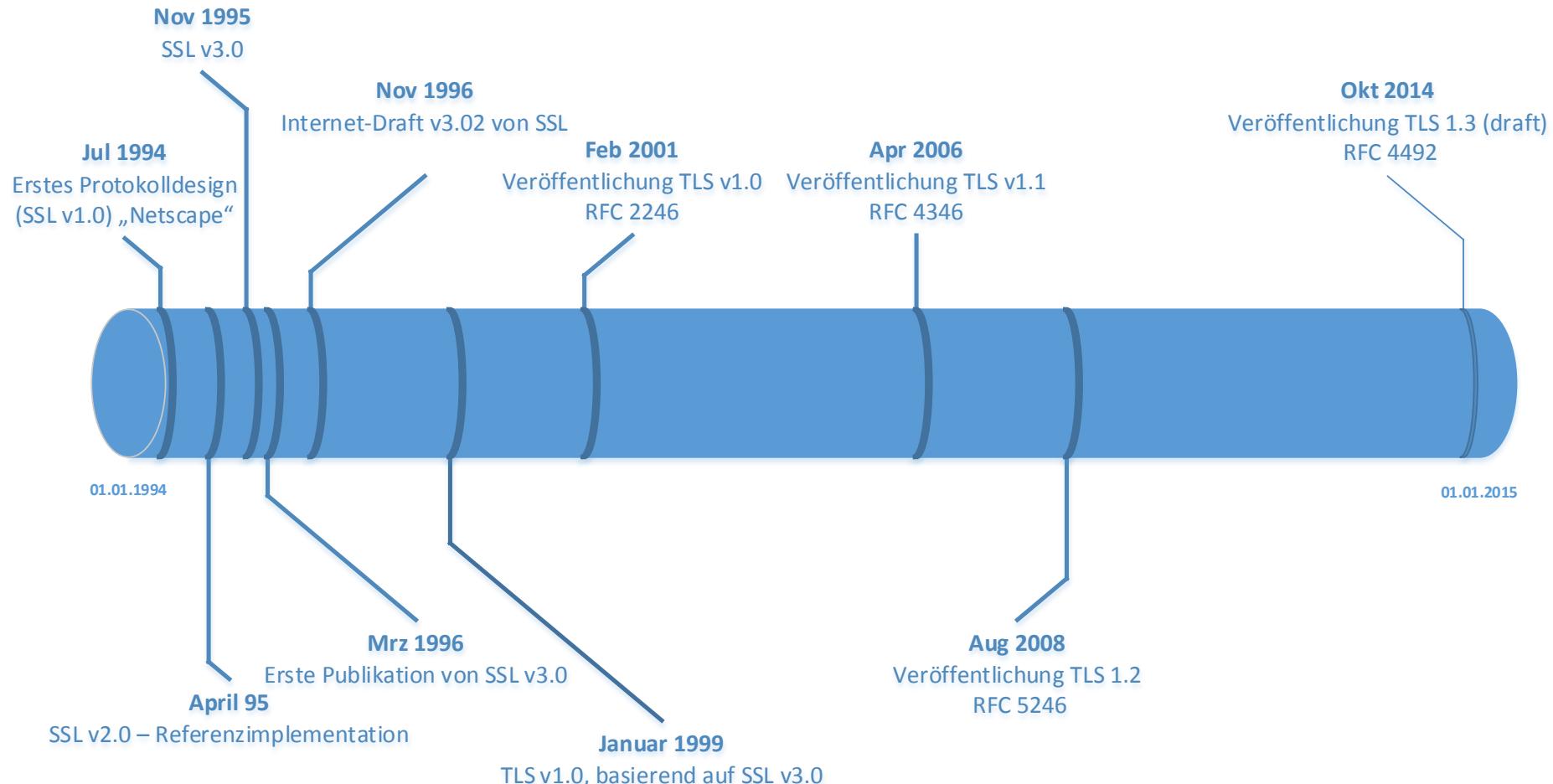
Transportsicherheit Secure Sockets Layer

Transportsicherheit

Grundlagen – und grobe Übersicht

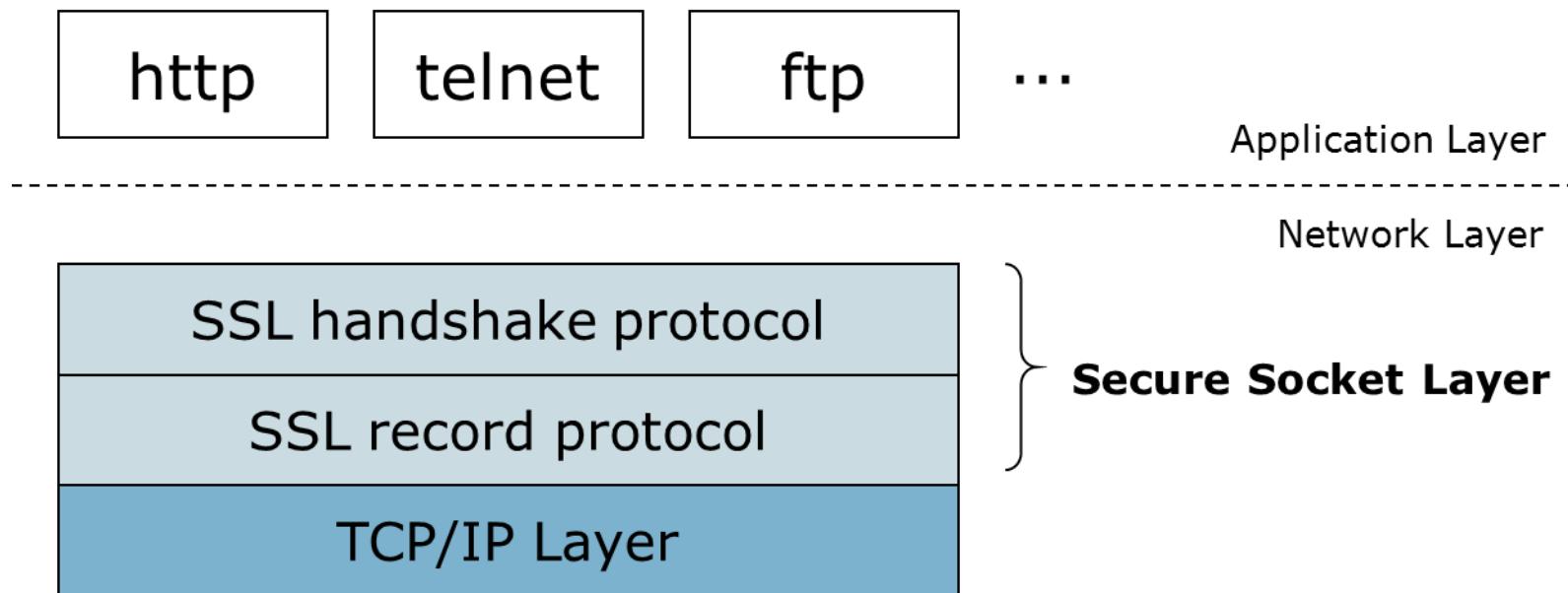
Java EE 7 Security – Transportsicherheit (SSL / TLS)

Secure Sockets Layer (SSL) resp. Transport Security Layer (TLS) Geschichte



Java EE 7 Security – Transportsicherheit (SSL / TLS)

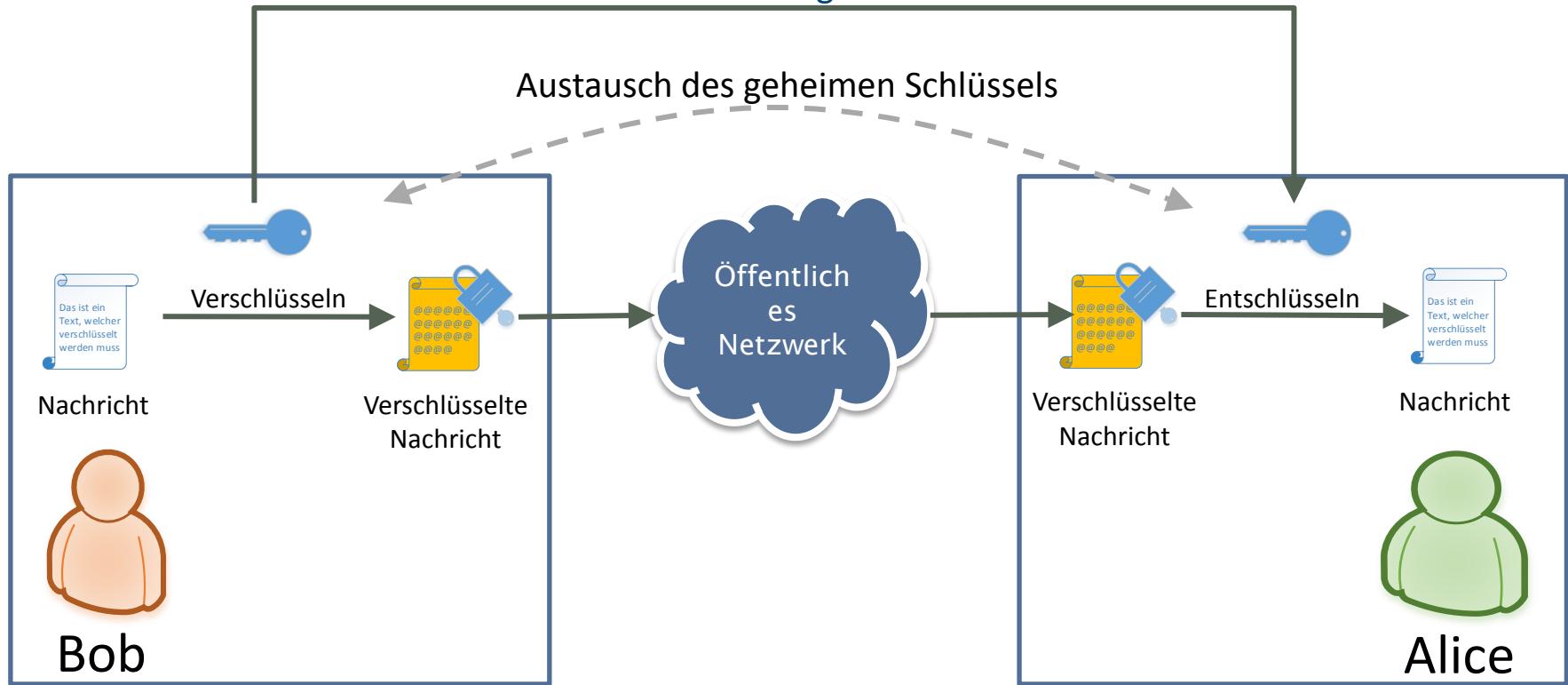
OSI Modell Layer 3 / 4 (Transport / Netzwerk)



Java EE 7 Security – Transportsicherheit (SSL / TLS)

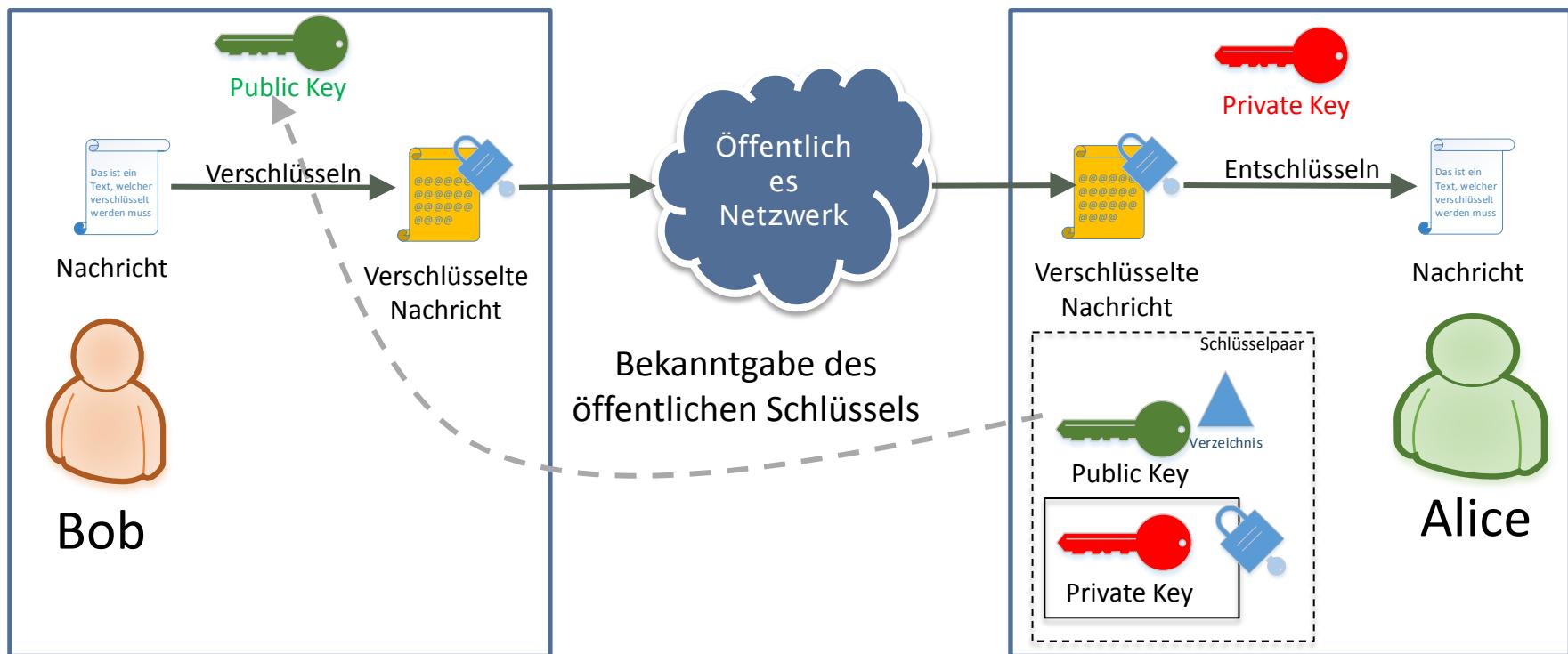
Symmetrisches Verschlüsselungs-Verfahren

Ein Schlüssel für die Ver- und
Entschlüsselung der Daten



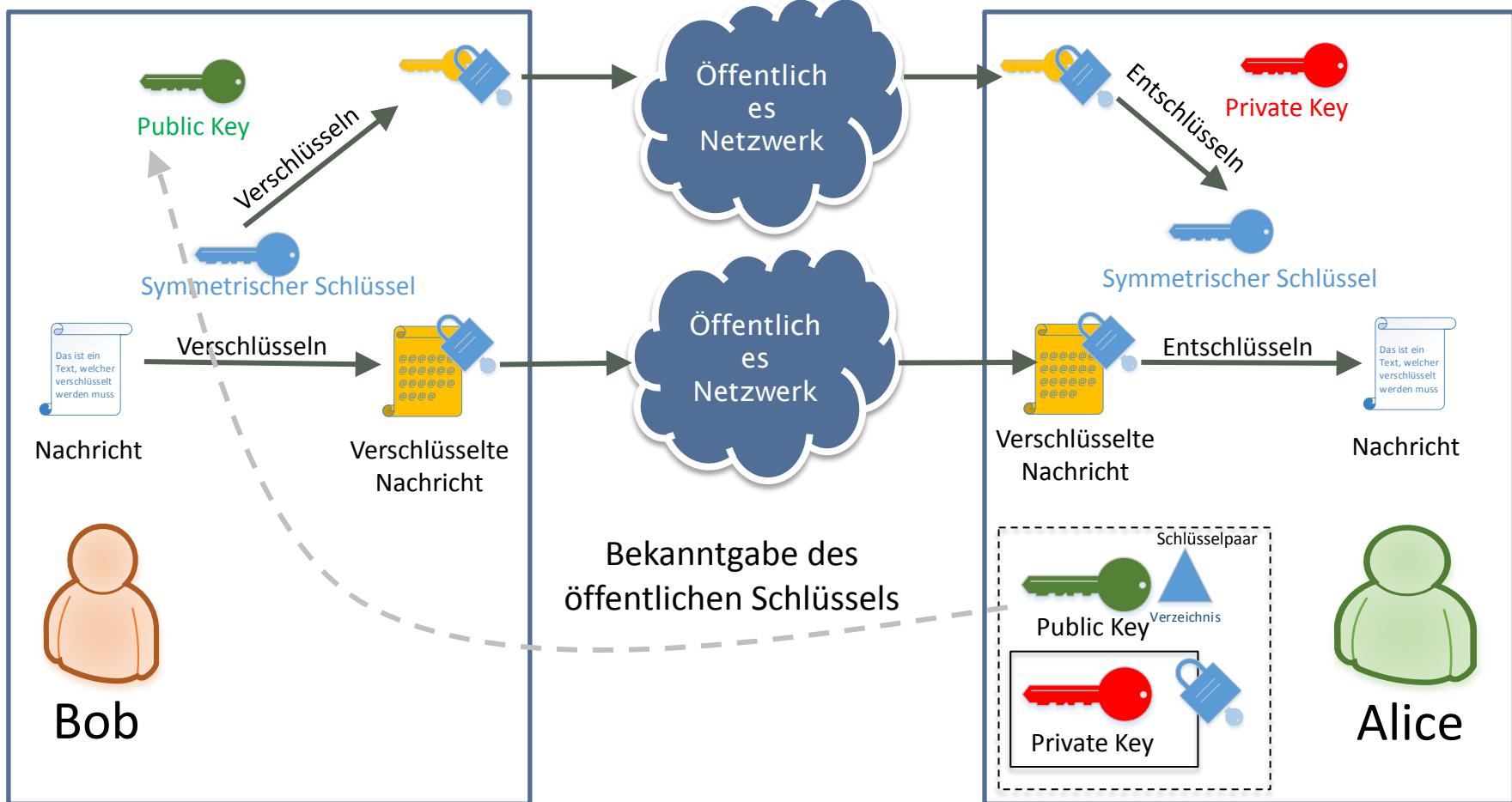
Java EE 7 Security – Transportsicherheit (SSL / TLS)

Asymmetrisches Verschlüsselungs-Verfahren

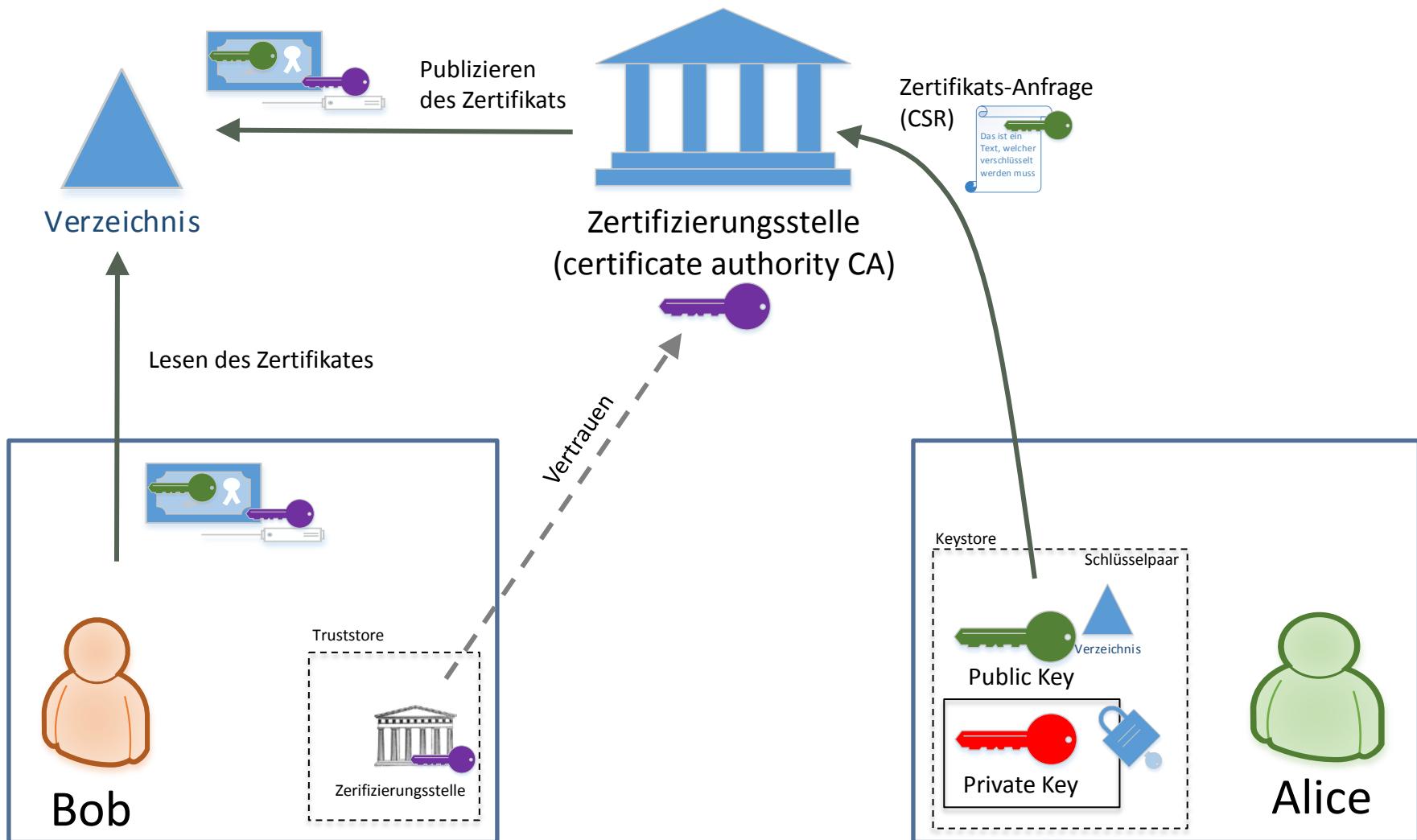


Java EE 7 Security – Transportsicherheit (SSL / TLS)

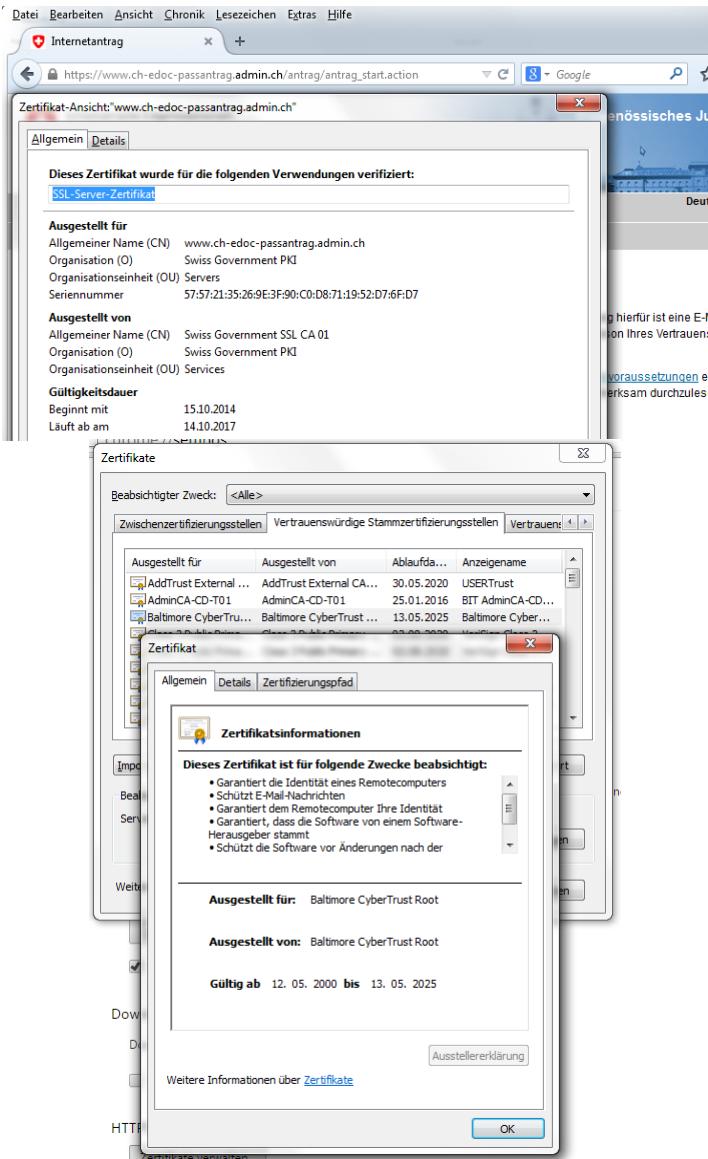
Hybrides Verschlüsselungs-Verfahren



Java EE 7 Security – Transportsicherheit (SSL / TLS)



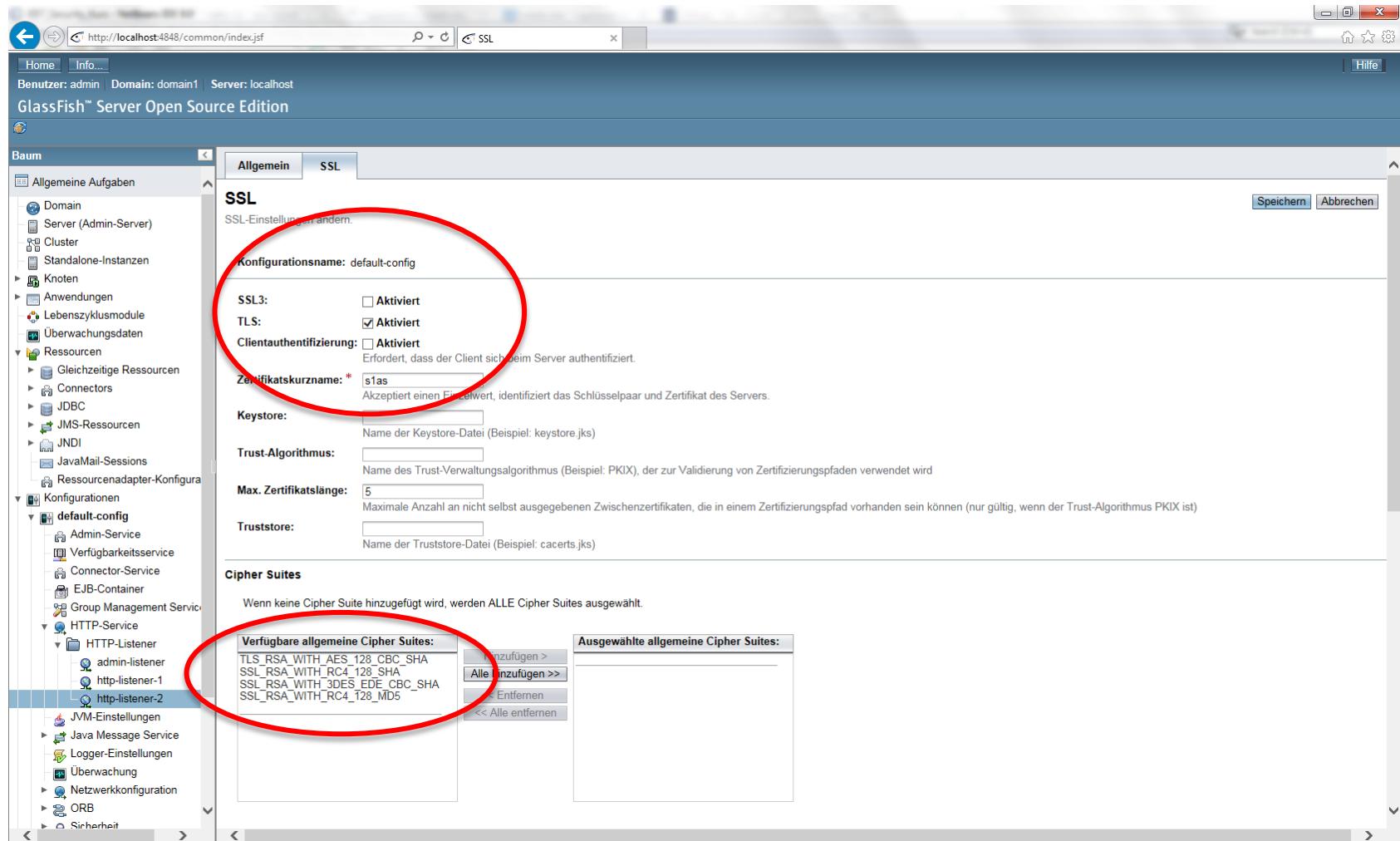
Java EE 7 Security – Transportsicherheit (SSL / TLS)



```
....Version:¶
.....Version·3¶
....Seriennummer:¶
.....57:57:21:35:26:9E:90:C0:D8:71:19:52:D7:6F:D7¶
....Zertifikatsunterzeichnungs-Algorithmus:¶
.....PKCS ·#1 ·SHA-256 ·mit ·RSA-Verschlüsselung¶
....Aussteller:¶
.....·CN ·= ·Swiss ·Government ·SSL ·CA ·01¶
.....·OU ·= ·Certification ·Authorities¶
.....·OU ·= ·Services¶
.....·O ·= ·Swiss ·Government ·PKI¶
.....·C ·= ·CH¶
....Validität:¶
.....Nicht ·vor:¶
.....·15.10.2014 ·16:50:29¶
.....(15.10.2014 ·14:50:29 ·GMT) ¶
.....Nicht ·nach:¶
.....·14.10.2017 ·16:50:29¶
.....(14.10.2017 ·14:50:29 ·GMT) ¶
....Inhaber:¶
.....·CN ·= ·www.ch-edoc-passantrag.admin.ch¶
.....·OU ·= ·SSL¶
.....·OU ·= ·Servers¶
.....·O ·= ·Swiss ·Government ·PKI¶
.....·C ·= ·CH¶
....Angaben ·zum ·öffentlichen ·Schlüssel ·des ·Inhabers:¶
.....·Public-Key-Algorithmus ·des ·Inhabers:¶
.....·PKCS ·#1 ·RSA-Verschlüsselung¶
....Öffentlicher ·Schlüssel ·des ·Inhabers:¶
.....Modulus ·(2048 ·Bits) :¶
.....·c3 ·c7 ·95 ·35 ·14 ·0d ·66 ·7a ·e0 ·75 ·51 ·e1 ·d2 ·57 ·4e ·b0 ·¶
.....·19 ·1f ·.....¶
.....Exponent ·(24 ·Bits) :¶
.....·65537¶
....Erweiterungen:¶
.....·.....¶
....Erweiterter ·Schlüsselgebrauch:¶
.....·TLS-Webserver-Authentifikation ·(1.3.6.1.5.5.7.3.1) ¶
Signaturwert ·des ·Zertifikats:¶
....Größe: ·512 ·Bytes ·/ ·4096 ·Bits¶
....·c7 ·4d ·9c ·bc ·6d ·87 ·cd ·52 ·f3 ·2a ·b6 ·13 ·a6 ·71 ·29 ·18 ·¶
....·d8 ·fd ·.....¶
```

Java EE 7 Security – Transportsicherheit (SSL / TLS)

Umsetzung im Glassfish (am Beispiel des Standard-Zertifikates «s1as»)



The screenshot shows the Glassfish Admin Console interface for managing SSL configurations. The left sidebar shows a tree view of the server configuration, with the 'SSL' tab selected in the top navigation bar. The main content area displays the 'SSL' configuration for the 'default-config' profile. A red circle highlights the 'SSL3' and 'TLS' sections, which are both checked as 'Aktiviert' (Enabled). Another red circle highlights the 'Cipher Suites' section, showing a list of available cipher suites and a list of selected cipher suites. The available cipher suites include: TLS_RSA_WITH_AES_128_CBC_SHA, SSL_RSA_WITH_RC4_128_SHA, SSL_RSA_WITH_3DES_EDE_CBC_SHA, and SSL_RSA_WITH_RC4_128_MD5.

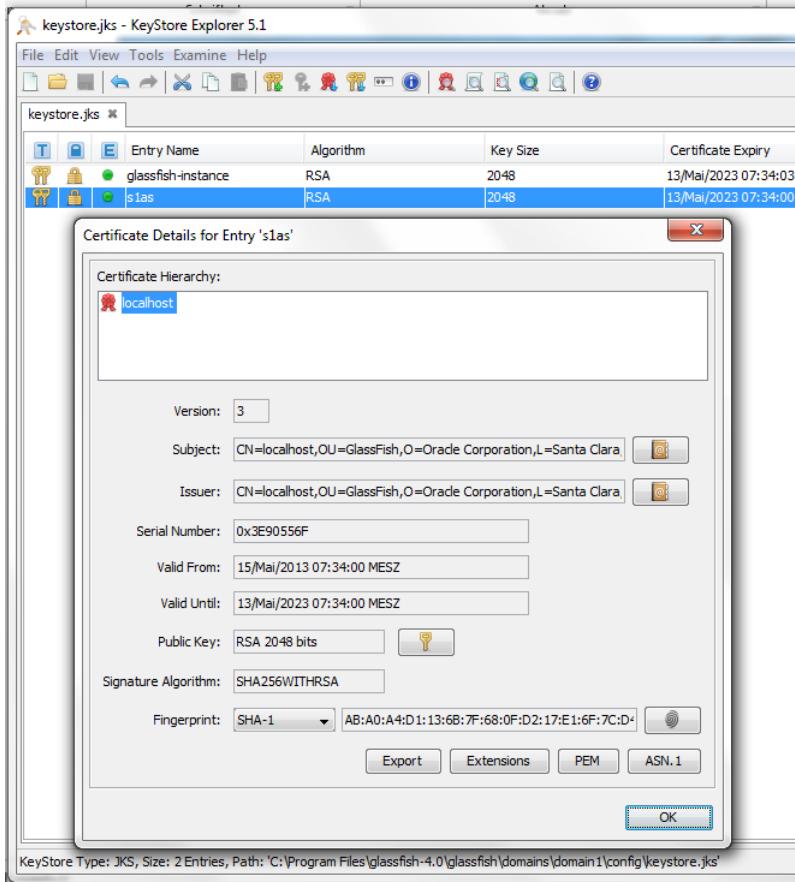
Verfügbarer allgemeine Cipher Suites:
TLS_RSA_WITH_AES_128_CBC_SHA
SSL_RSA_WITH_RC4_128_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_RC4_128_MD5

Ausgewählte allgemeine Cipher Suites:

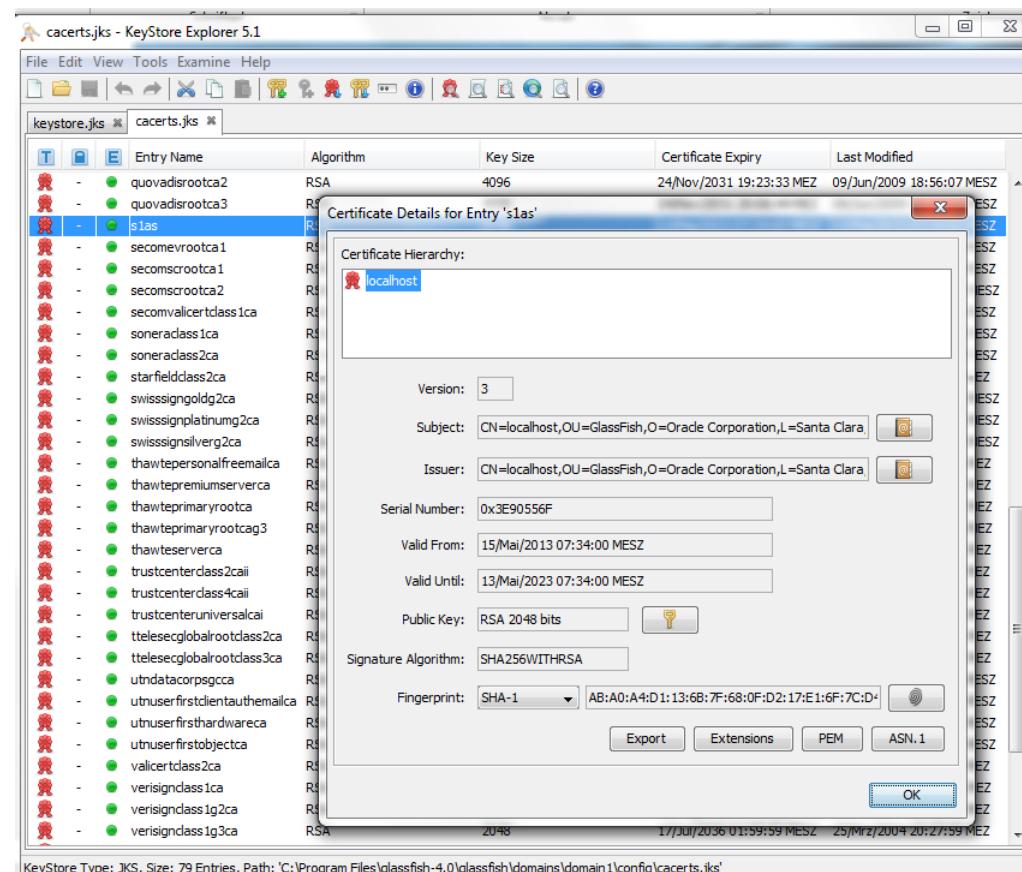
Java EE 7 Security – Transportsicherheit (SSL / TLS)

Umsetzung im Glassfish (am Beispiel des Standard-Zertifikates «s1as»)

Glassfish Keystore

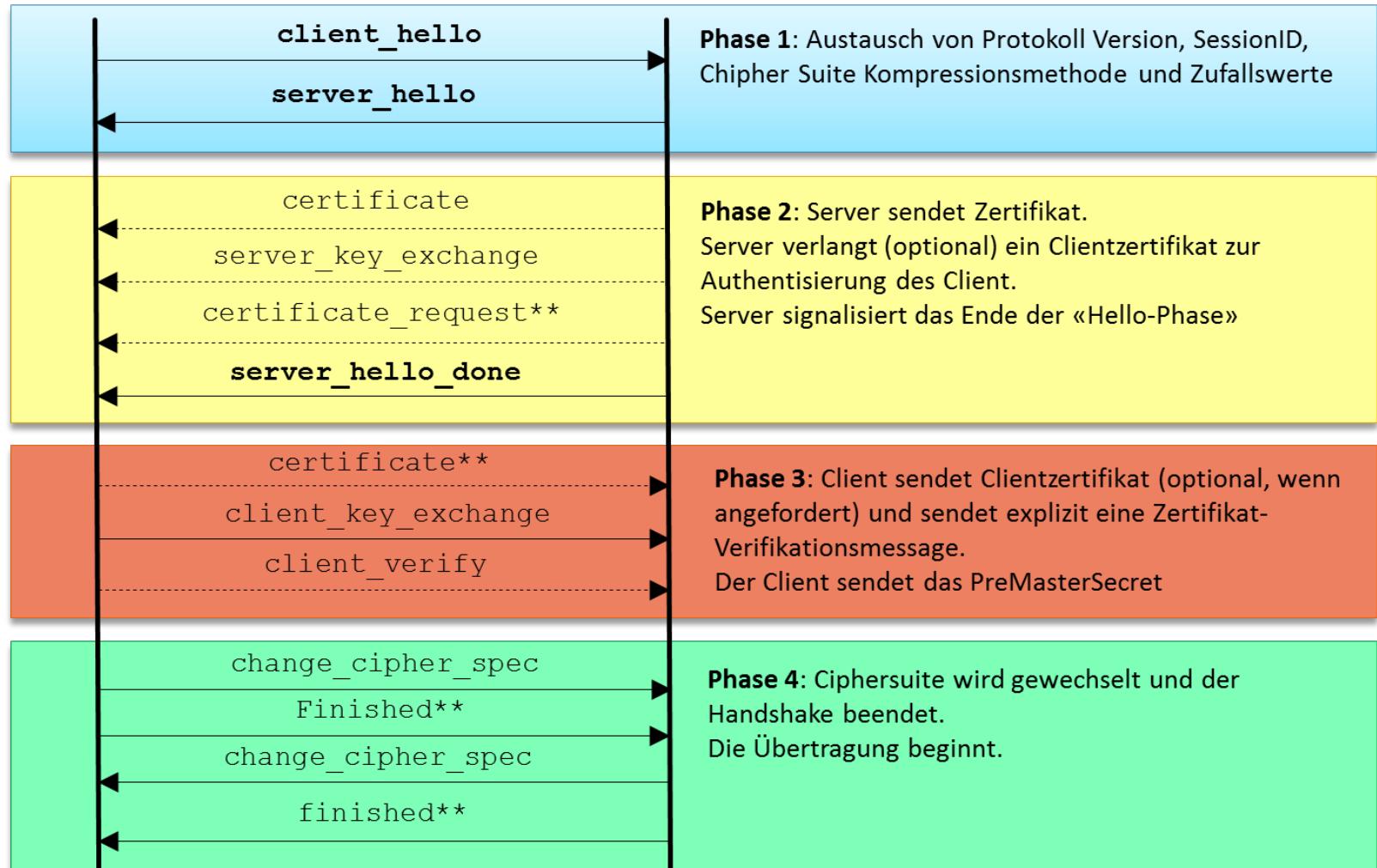


Glassfish Truststore



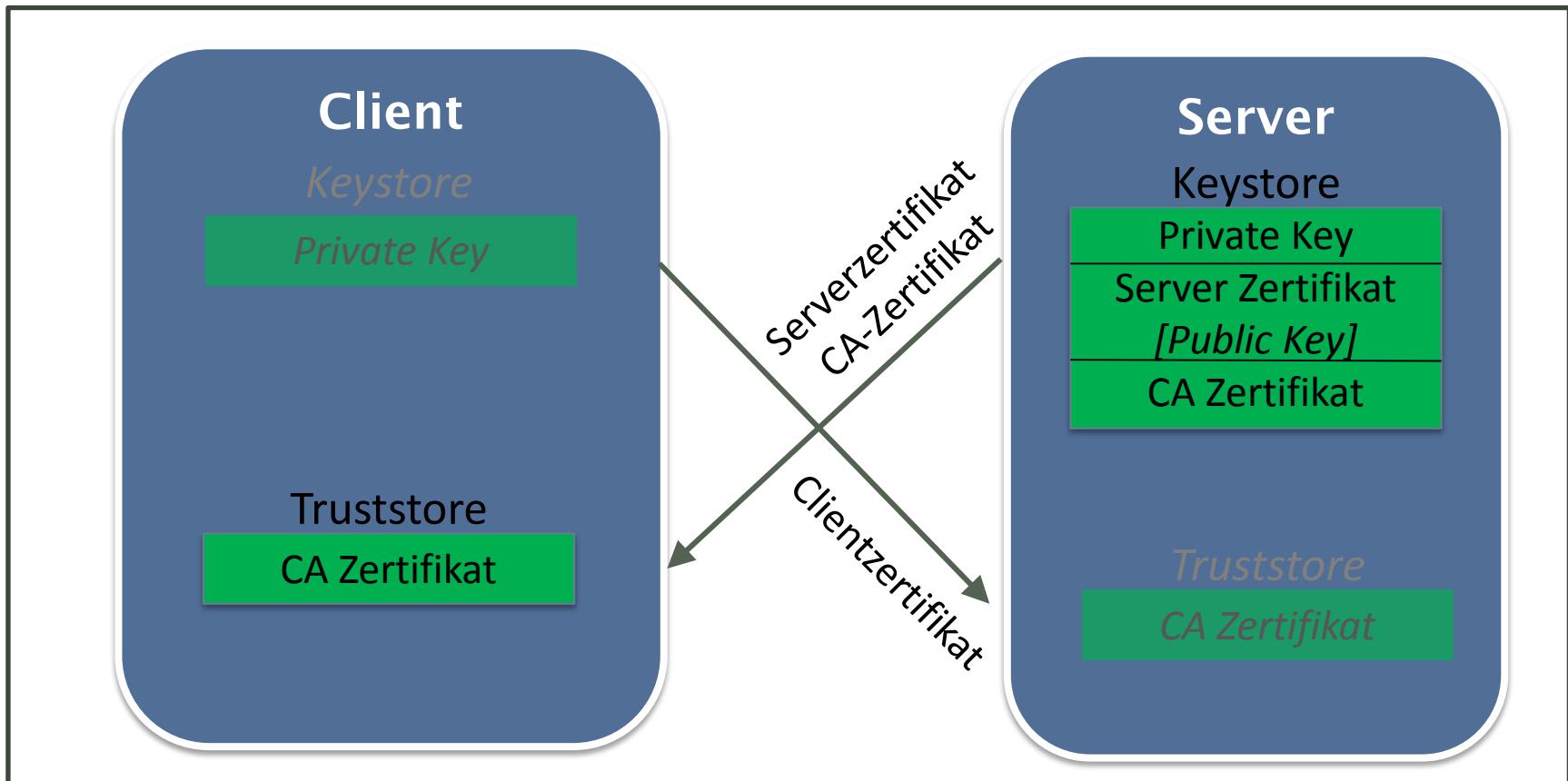
Java EE 7 Security – Transportsicherheit (SSL / TLS)

SSL-Handshake Protokoll



Java EE 7 Security – Transportsicherheit (SSL / TLS)

Keystore und Truststore



Java EE 7 Security – Transportsicherheit (SSL / TLS)

Deklarative Konfiguration im Web-Container (web.xml)

Listing: security-constraint im web.xml

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>SecureServlet</web-resource-name>
        <description>Matches all pages from protected</description>
        <url-pattern>/protected/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>user</role-name>
        <role-name>administrator</role-name>
    </auth-constraint>
    <user-data-constraint>
        <description>SSL required</description>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Definition der Transport-Sicherheit

Java EE 7 Security – Transportsicherheit (SSL / TLS)

Deklarative (propriätäre) Konfiguration im EJB-Container (glassfish-ejb-jar.xml)

Listing: Konfiguration im glassfish-ejb-jar.xml

```
<enterprise-beans>
  <ejb>
    <ejb-name>SecureWebService</ejb-name>
    <webservice-endpoint>
      <port-component-name>Service</port-component-name>
      <login-config>
        <auth-method>BASIC</auth-method>
        <realm>SecureRealm</realm>
      </login-config>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </webservice-endpoint>
  </ejb>
```

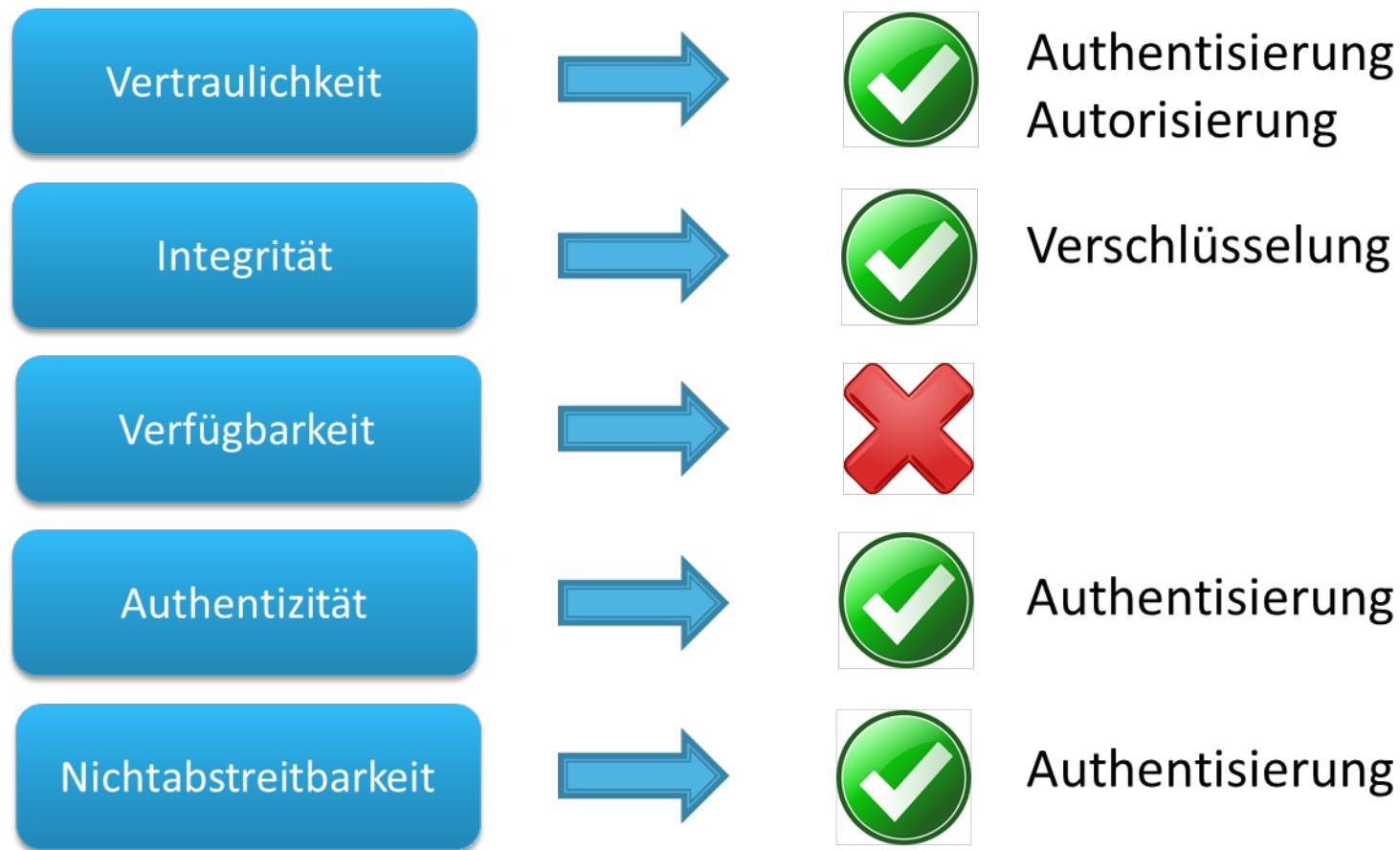
Definition der Transport-Sicherheit

Java EE 7 Security – Transportsicherheit (SSL / TLS)

Pitfalls im Umgang mit SSL

- Wenn einmal auf SSL [HTTPs] gewechselt wurde, darf nicht mehr auf HTTP umgeschaltet werden können (Cookies können gesniffed werden).
- Die Standardeinstellungen im Applikationsserver müssen zwingend überprüft werden auf:
 - Aktualität und Support der neusten Protokolle (TLS 1.2)
 - Aktualität der Cipher-Suites
- Keystores müssen gut gesichert sein - am Besten in einem HSM (Hardware-Security-Modul)

Java EE 7 Security – Transportsicherheit (SSL / TLS)



Java EE 7 Security: Übung 5 Transportsicherheit

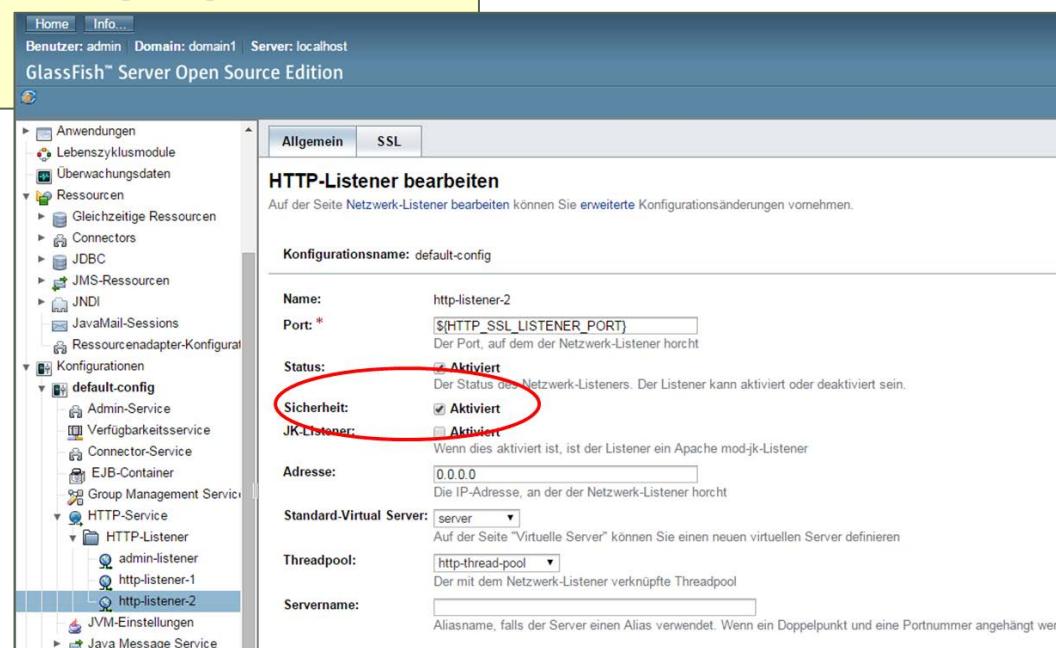
Einschalten von SSL für die ganze (Web-) Applikation

- Importieren des «JEE7_Security_Kurs_Uebung_4_Loesung.zip» Projektes in Netbeans
- Erstellen einer `<security-constraint>` für
 - Beliebige Benutzer
 - Alle Methoden
 - Alle Ressourcen innerhalb der Web-Applikation (URL Pattern = `/*`)
- Die Lösung befindet sich im Projekt
«JEE7_Security_Kurs_Uebung_5_Loesung.zip»
- **Zusatzfrage:** Gibt es mehrere Möglichkeiten?

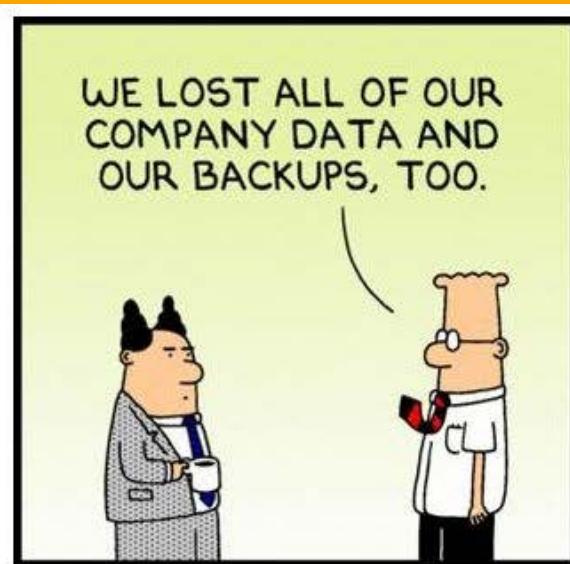
Java EE 7 Security: Übung 5 Transportsicherheit

```
<!-- Definition der Constraint für das SecureServlet -->
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Root</web-resource-name>
        <description>Matches all pages from protected</description>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <description>SSL required</description>
        <transport-guarantee>CONFIDENTIAL </transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Home | Info...
Benutzer: admin | Domain: domain1 | Server: localhost
GlassFish™ Server Open Source Edition



Besten Dank – viel Erfolg im Umgang mit Security



Transportsicherheit

Zusatzaufgaben (Erstellen von Schlüsselpaaren und Zertifikaten)

Java EE 7 Security – Transportsicherheit (SSL / TLS)

Erstellen von Schlüsselpaaren und eines Zertifikates (selfsigned) für einen WebServer

- ▶ "%JAVA_HOME%/bin/keytool.exe" -genkey -alias WebServerCertificate -keyalg RSA -keypass changeit -storepass changeit -keystore "%GLASSFISH_HOME%/glassfish/domains/domain1/config/keystore.jks"
- ▶ Wie lautet Ihr Vor- und Nachname? [localhost]: [localhost]
- ▶ Wie lautet der Name Ihrer organisatorischen Einheit? [BFH]
- ▶ Wie lautet der Name Ihrer Organisation? [SWS]
- ▶ Wie lautet der Name Ihrer Stadt oder Gemeinde? [Bern]
- ▶ Wie lautet der Name Ihres Bundeslands? [Unknown]
- ▶ Wie lautet der Ländercode (zwei Buchstaben) für diese Einheit? [CH]: [CH]

Java EE 7 Security – Transportsicherheit (SSL / TLS)

► **Export des WebServer-Zertifikates**

```
"%JAVA_HOME%\bin\keytool.exe" -export -alias WebServerCertificate -storepass  
changeit -file c:\temp\webserver.cer -keystore  
"%GLASSFISH_HOME%\glassfish\domains\domain1\config\keystore.jks"
```

► **Erstellen eines Certificate-Signing Request**

```
"%JAVA_HOME%\bin\keytool.exe" -certreq -keyalg RSA -alias WebServerCertificate -  
file c:\temp\certreq.txt -keystore  
"%GLASSFISH_HOME%\glassfish\domains\domain1\config\keystore.jks"
```

► **Versenden des Certificate-Signing Request an eine CA**

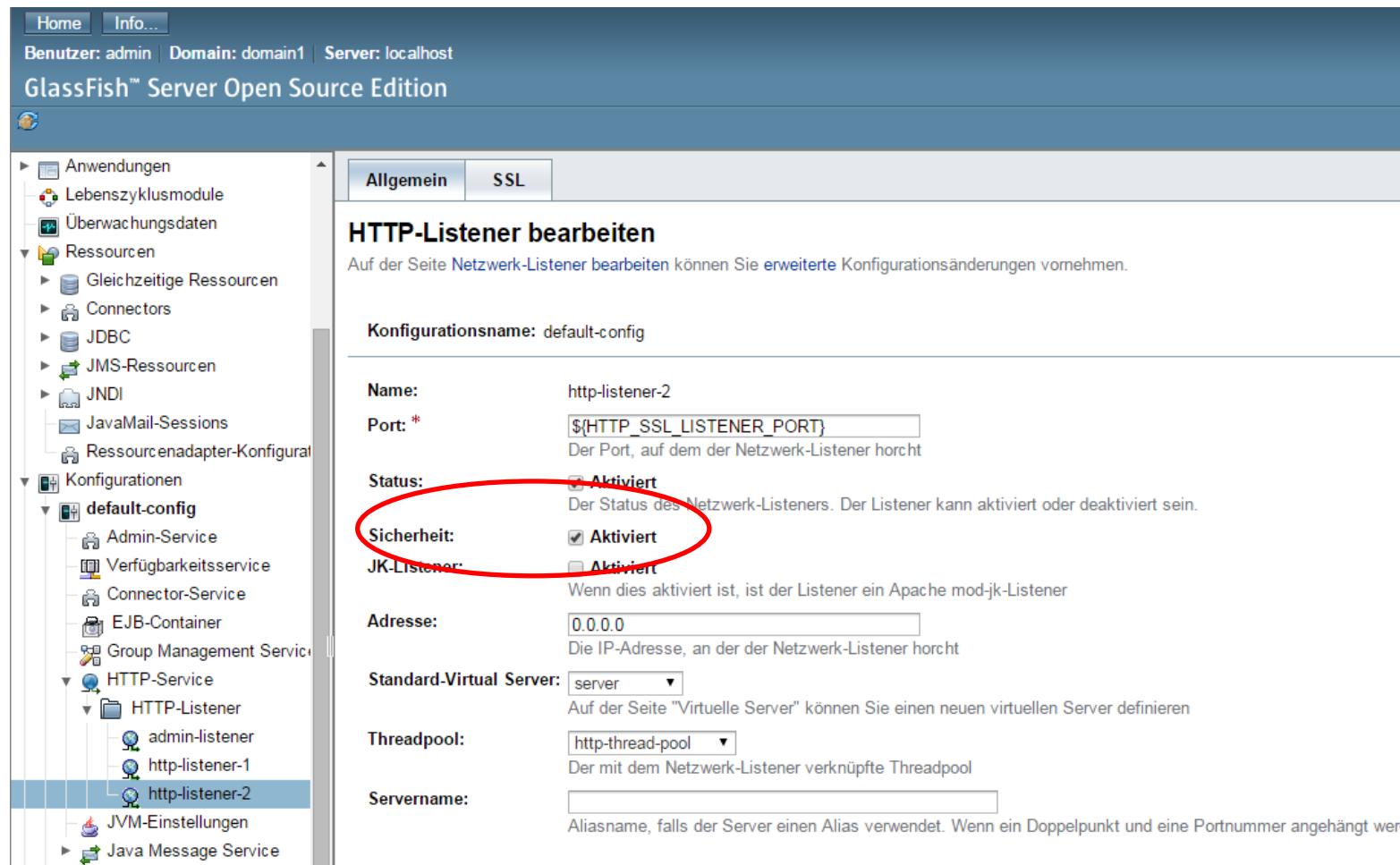
► An dieser Stelle stellt eine autorisierte CA ein entsprechendes Server-Zertifikat aus

Java EE 7 Security – Transportsicherheit (SSL / TLS)

- ▶ **Import des (Root) WebServer-Zertifikates (z.B. ausgestellt von der CA)**
- ▶ "%JAVA_HOME%\bin\keytool.exe" -import -v -trustcacerts -alias WebServerCertificate -file c:\Temp\webserverCA.pem -keystore "%GLASSFISH_HOME%\glassfish\domains\domain1\config\cacerts.jks" -keypass changeit -storepass changeit

Java EE 7 Security – Transportsicherheit (SSL / TLS)

Konfiguration von SSL im Glassfish 4.x



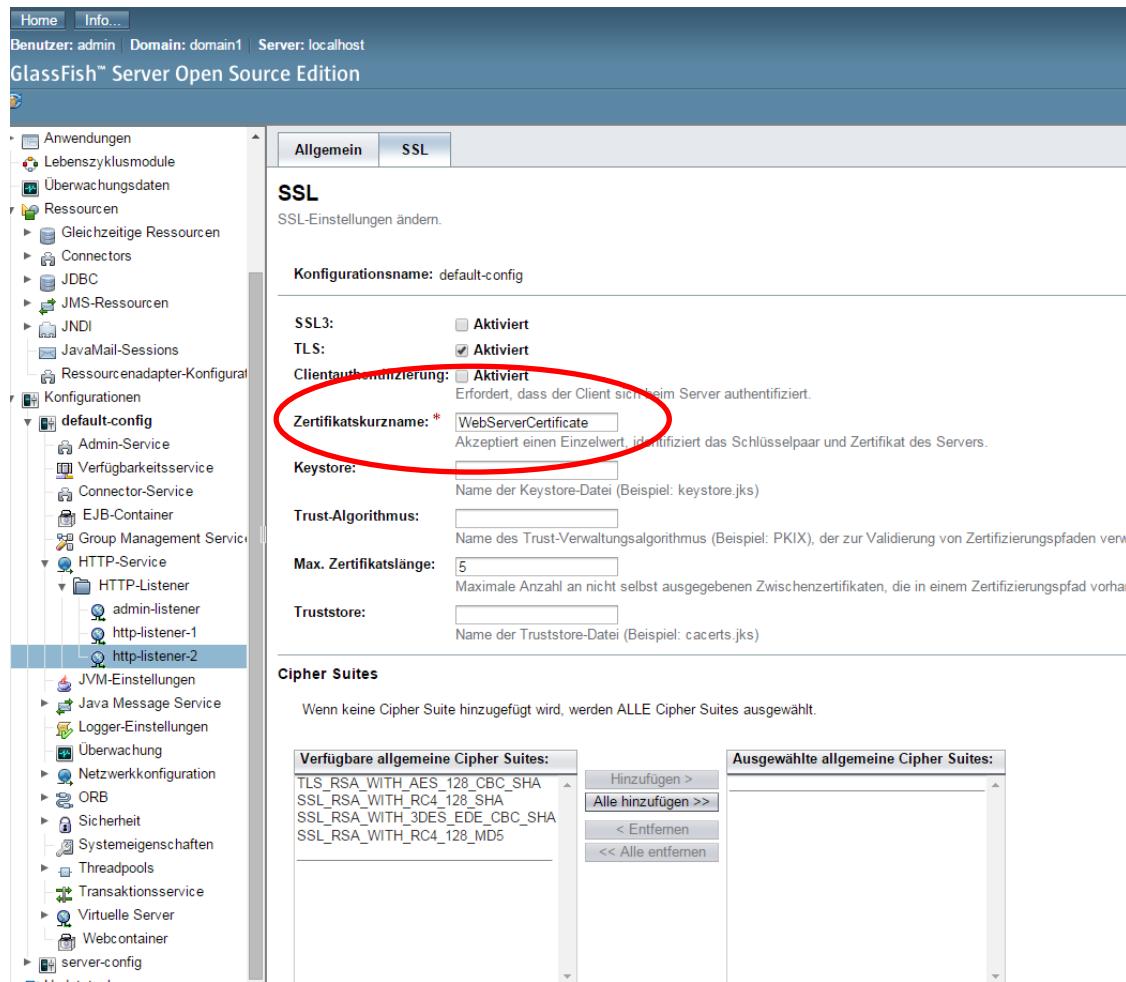
The screenshot shows the Glassfish 4.x Administration Console interface. The left sidebar navigation tree includes 'Anwendungen', 'Lebenszyklusmodule', 'Überwachungsdaten', 'Ressourcen' (selected), 'JNDI', 'JavaMail-Sessions', 'Ressourcenadapter-Konfigurationen', 'Konfigurationen' (selected), 'default-config' (selected), 'HTTP-Service' (selected), 'HTTP-Listener' (selected), 'admin-listener', 'http-listener-1', 'http-listener-2' (selected), 'JVM-Einstellungen', and 'Java Message Service'. The top navigation bar shows 'Home' and 'Info...' buttons, and the status 'Benutzer: admin | Domain: domain1 | Server: localhost'. The title 'GlassFish™ Server Open Source Edition' is displayed. The main content area is titled 'HTTP-Listener bearbeiten' and displays configuration for 'http-listener-2'. The configuration includes:

- Konfigurationsname:** default-config
- Name:** http-listener-2
- Port:** * \${HTTP_SSL_LISTENER_PORT} (Der Port, auf dem der Netzwerk-Listener horcht)
- Status:** Aktiviert (Der Status des Netzwerk-Listeners. Der Listener kann aktiviert oder deaktiviert sein.)
- Sicherheit:** Aktiviert (Sicherheit)
- JK-Listener:** Aktiviert (Wenn dies aktiviert ist, ist der Listener ein Apache mod-jk-Listener)
- Adresse:** 0.0.0.0 (Die IP-Adresse, an der der Netzwerk-Listener horcht)
- Standard-Virtual Server:** server (Auf der Seite "Virtuelle Server" können Sie einen neuen virtuellen Server definieren)
- Threadpool:** http-thread-pool (Der mit dem Netzwerk-Listener verknüpfte Threadpool)
- Servername:** (Aliasname, falls der Server einen Alias verwendet. Wenn ein Doppelpunkt und eine Portnummer angehängt wird)

Two fields, 'Sicherheit' and 'JK-Listener', are circled in red.

Java EE 7 Security – Transportsicherheit (SSL / TLS)

Konfiguration von SSL im Glassfish 4.x



The screenshot shows the Glassfish 4.x Administration Console with the following details:

- Header:** Home, Info..., Benutzer: admin, Domain: domain1, Server: localhost, GlassFish™ Server Open Source Edition.
- Left Sidebar:** Anwendungen, Lebenszyklusmodule, Überwachungsdaten, Ressourcen (Gleichzeitige Ressourcen, Connectors, JDBC, JMS-Ressourcen, JNDI, JavaMail-Sessions, Ressourcenadapter-Konfiguration), Konfigurationen, default-config (Admin-Service, Verfügbarkeitsservice, Connector-Service, EJB-Container, Group Management Service, HTTP-Service, HTTP-Listener, admin-listener, http-listener-1, http-listener-2, JVM-Einstellungen, Java Message Service, Logger-Einstellungen, Überwachung, Netzwerkkonfiguration, ORB, Sicherheit, Systemeigenschaften, Threadpools, Transaktionsservice, Virtuelle Server, Webcontainer, server-config).
- SSL Tab:** Allgemein (selected), SSL.
- SSL Configuration:**
 - Konfigurationsname:** default-config
 - SSL3:** Aktiviert
 - TLS:** Aktiviert
 - Clientauthentifizierung:** Aktiviert
 - Zertifikatskurzname:** * (highlighted with a red circle)
 - Keystore:**
 - Trust-Algorithmus:**
 - Max. Zertifikatlänge:** 5
 - Truststore:**
- Cipher Suites:** A section for selecting cipher suites. It shows a list of available cipher suites on the left and a selected list on the right. The available list includes:
 - TLS_RSA_WITH_AES_128_CBC_SHA
 - SSL_RSA_WITH_RC4_128_SHA
 - SSL_RSA_WITH_3DES_EDE_CBC_SHA
 - SSL_RSA_WITH_RC4_128_MD5Buttons for managing the list include: Hinzufügen >, Alle hinzufügen >, < Entfernen, << Alle entfernen.

Java EE 7 Security –Übung Bookstore: Ziele

► **Vertraulichkeit**

- Users des Bookstore müssen authentisiert und autorisiert werden, um gewisse Funktionen auszuführen und Daten einzusehen

► **Integrität**

- Die Kommunikation ist zu schützen

► **Security-Roles**

- Erstellen eines **Rollen und Berechtigungskonzeptes** für den Administrator, die Customer und die Employees.
- Der Admin kann Employees erfassen (Admin GUI)
- Employees können Kunden verwalten (REST-API)
- Kunden können sich registrieren und ihre Daten (Accounts, Orders) verwalten (REST-API)
- Alle können Bücher suchen und den shopping cart verwenden (ohne Authentisierung)

Java EE 7 Security -Übung Bookstore: Design

- Erstellen eines Security Design für den Bookstore anhand des Berechtigungskonzeptes
- Aufzeigen im Komponenten-Modell wo bereits Security eingebaut worden sind (nicht JEE7 Security) und wo man noch Security Aspekte einbauen muss

