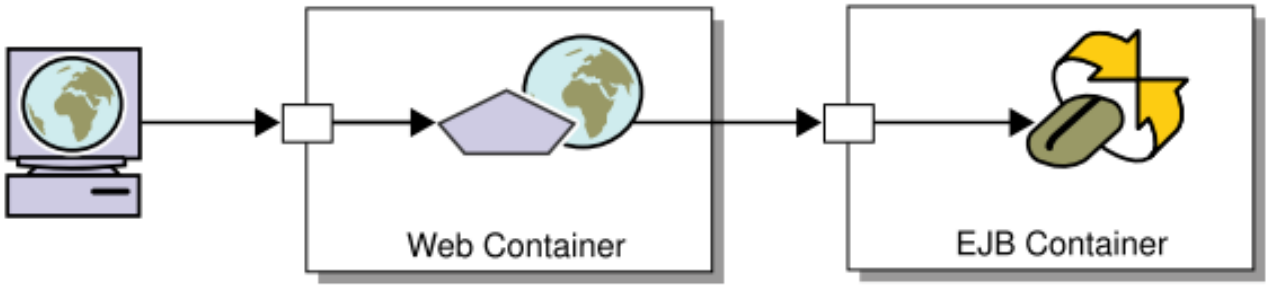
Security

The security model in the Java EE platform is primarily an *authorization* model.

- If required, the container authenticates the client.
- The container checks a client's rights to carry out the requested action on a component.
- After authorization is complete, the container invokes application code.

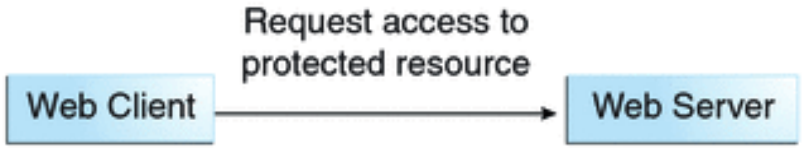


Quelle: Developing Applications for the Java™ EE Platform , FJ-310-EE5, oracle

Modul 15
Security Walkthrough

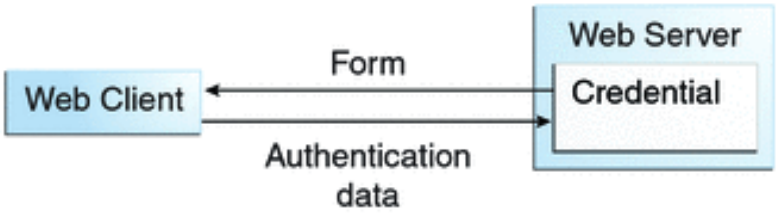
Step 1: Initial Request

Ich will zugreifen auf …



Step 2: Initial Authentication

Wer bin ich …



Credential
(dt.: „Berechtigungsnachweis“)

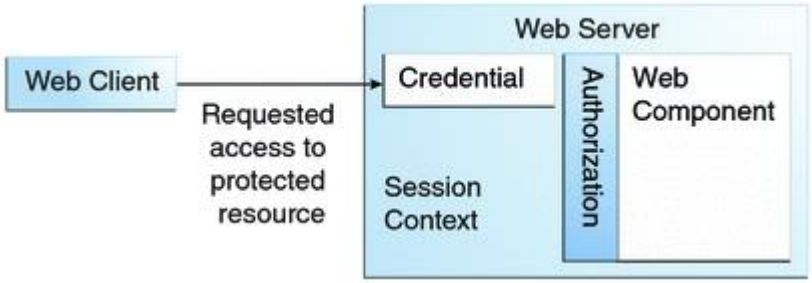
z.B. „gast“ oder „admin“ …

Quelle: Developing Applications for the Java™ EE Platform , FJ-310-EE5, oracle

Modul 15
Security Walkthrough

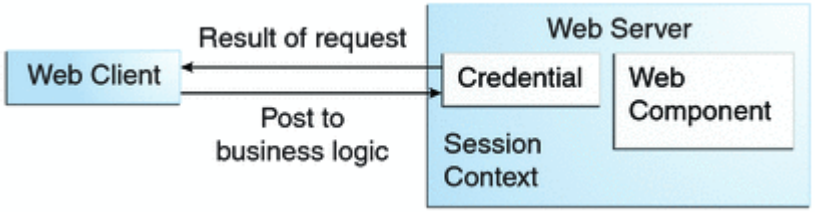
Step 3: URL Authorization

Bin ich autorisiert …



Step 4: Fulfilling the Original Request

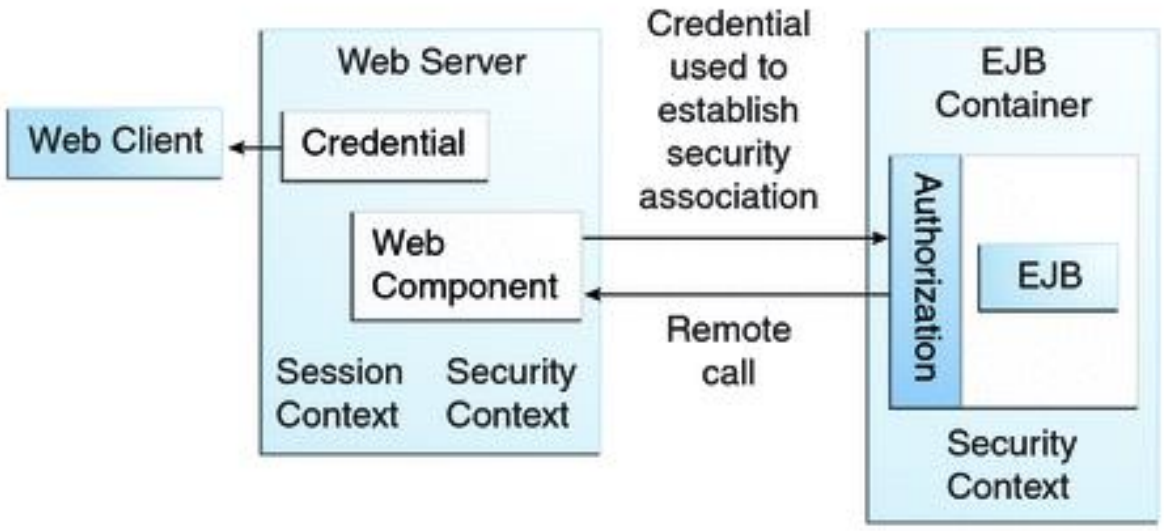
Zugriff auf die Web-Ressource …



Quelle: Developing Applications for the Java™ EE Platform , FJ-310-EE5, oracle

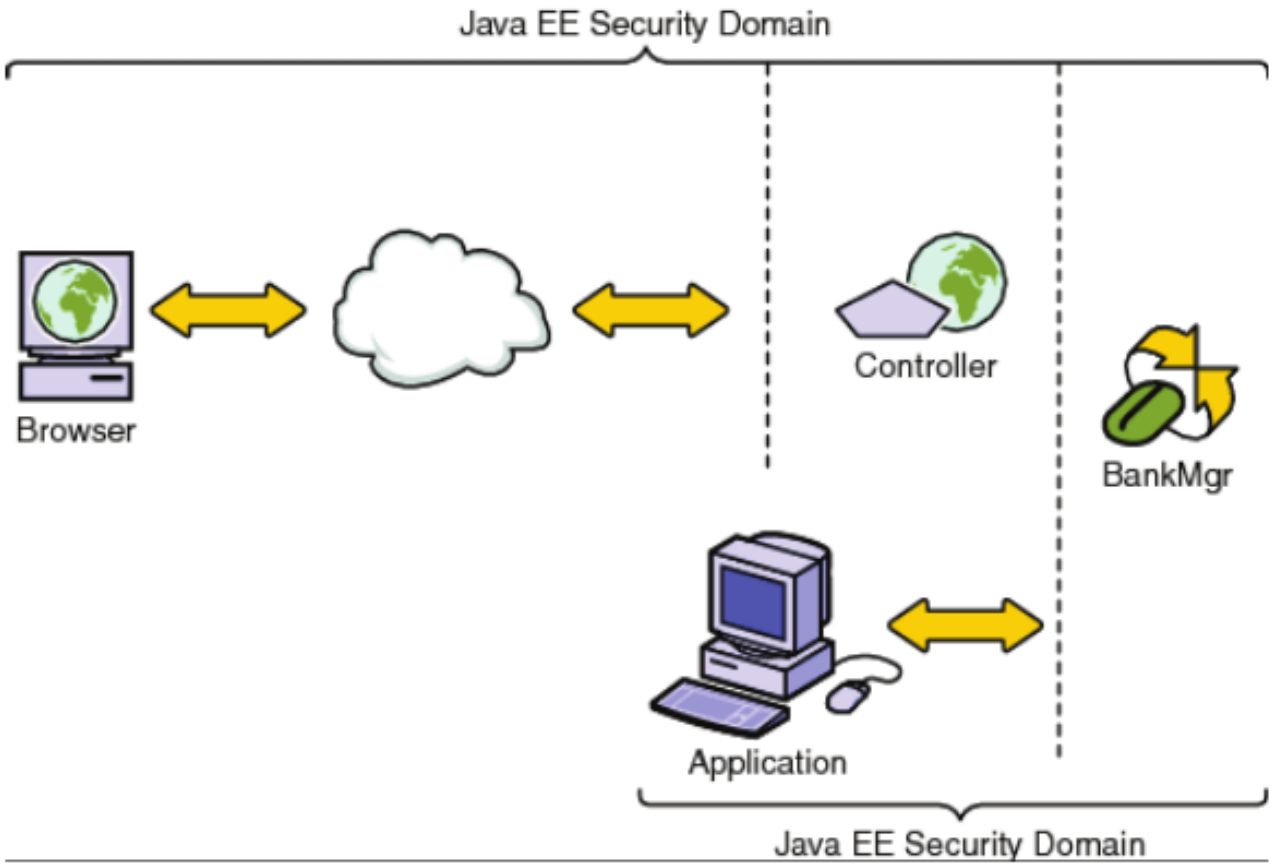
Step 5: Invoking Enterprise Bean Business Methods

Zugriff auf die EJB-Ressource …

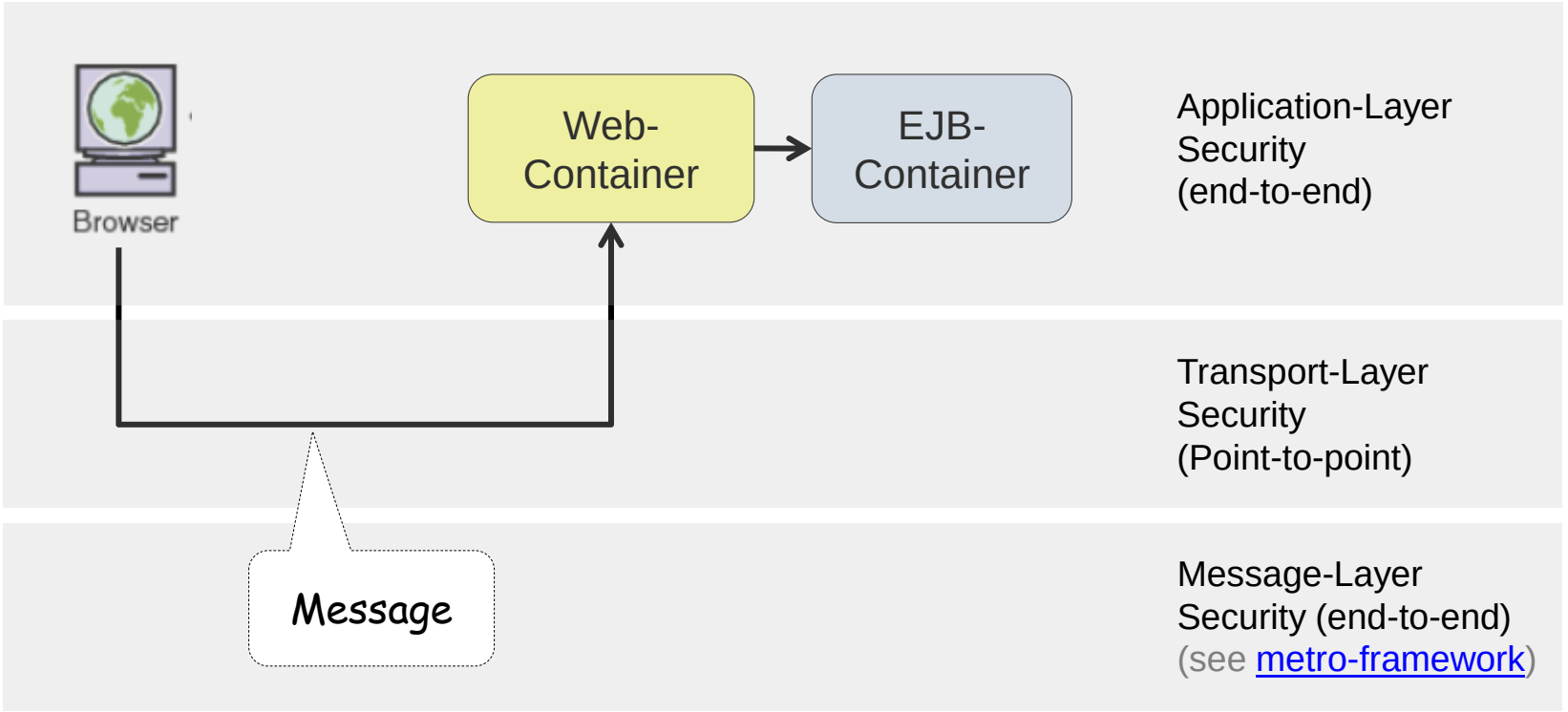


Quelle: Developing Applications for the Java™ EE Platform , FJ-310-EE5, oracle

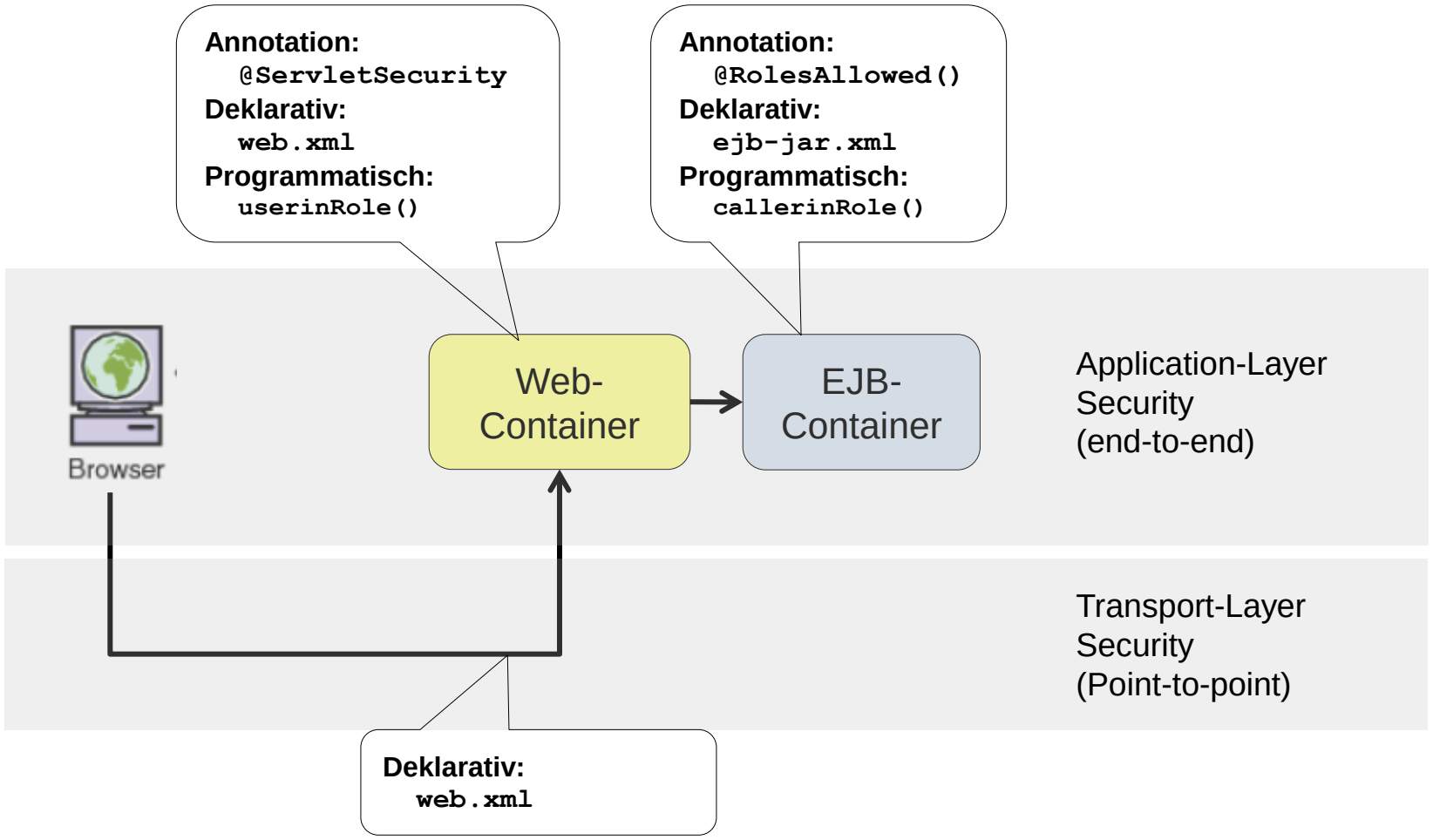
Modul 15
End-to-end security



Quellehttp://docs.oracle.com/javaee/6/tutorial/doc/

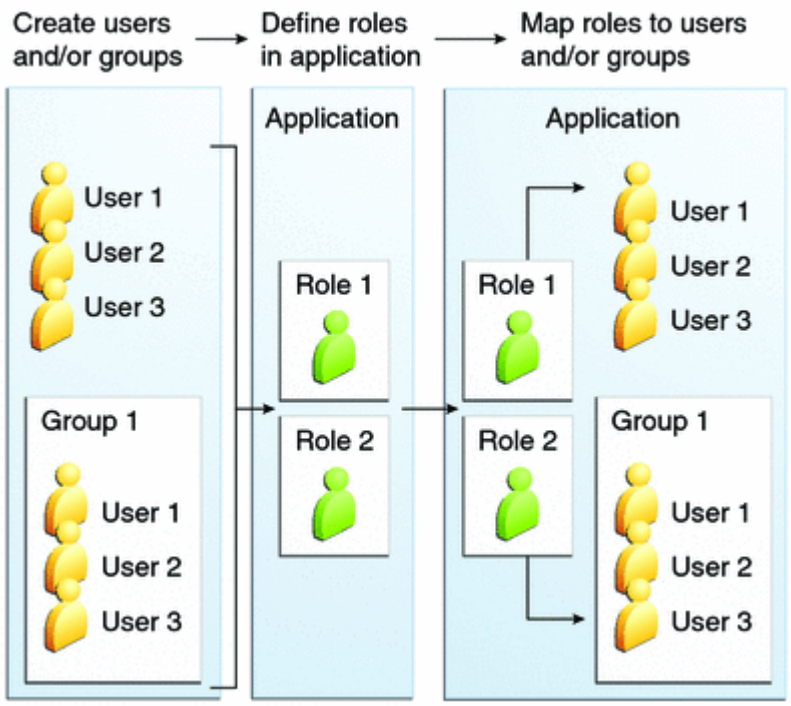

Browser
Web-
Container
EJB-
Container
Application-Layer
Security
(end-to-end)
Transport-Layer
Security
(Point-to-point)
Message
Message-Layer
Security (end-to-end)
(see metro-framework)

Modul 15
Securing Containers



Modul 15
Working with Realms, Users, Groups, and Roles

A **realm** is a security policy domain defined for a web or application server (user & groups). E.g. **`file-reaml, jdbc-realm, ldap-reaml,`** …



A **user** is an individual or application program identity that has been defined in the GlassFish Server. E.g.: **`hg@web.de`**

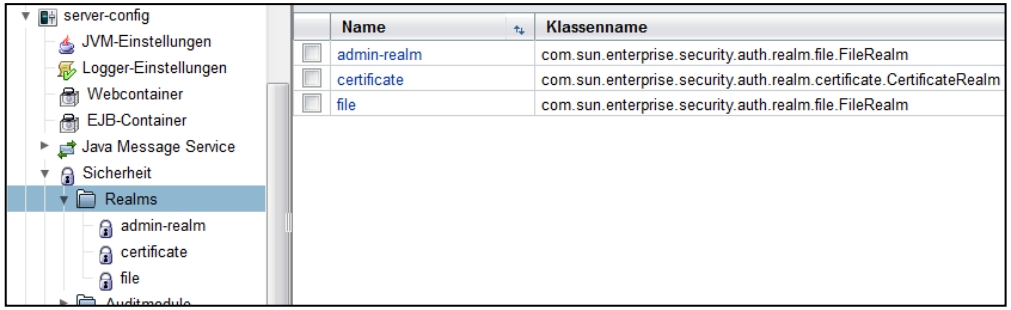
A **group** is a set of authenticated users (…).
E.g.: **`ai-studs, winp-studs`**

A **role** is an abstract name for the permission to access a particular set of resources in an application.
E.g.: **`studs, admin, user`**

Quellehttp://docs.oracle.com/javaee/6/tutorial/doc/

Modul 15
Managing Users and Groups on the GlassFish Server

Glassfish: Managing Users



Glassfish file-realm:
http://docs.oracle.com/javaee/6/tutorial/doc/bnbxj.html#bnbxr

JDBC-realm:
http://java.dzone.com/articles/jdbc-realm-and-form-based

Glassfish: Mapping Roles to Users and Groups

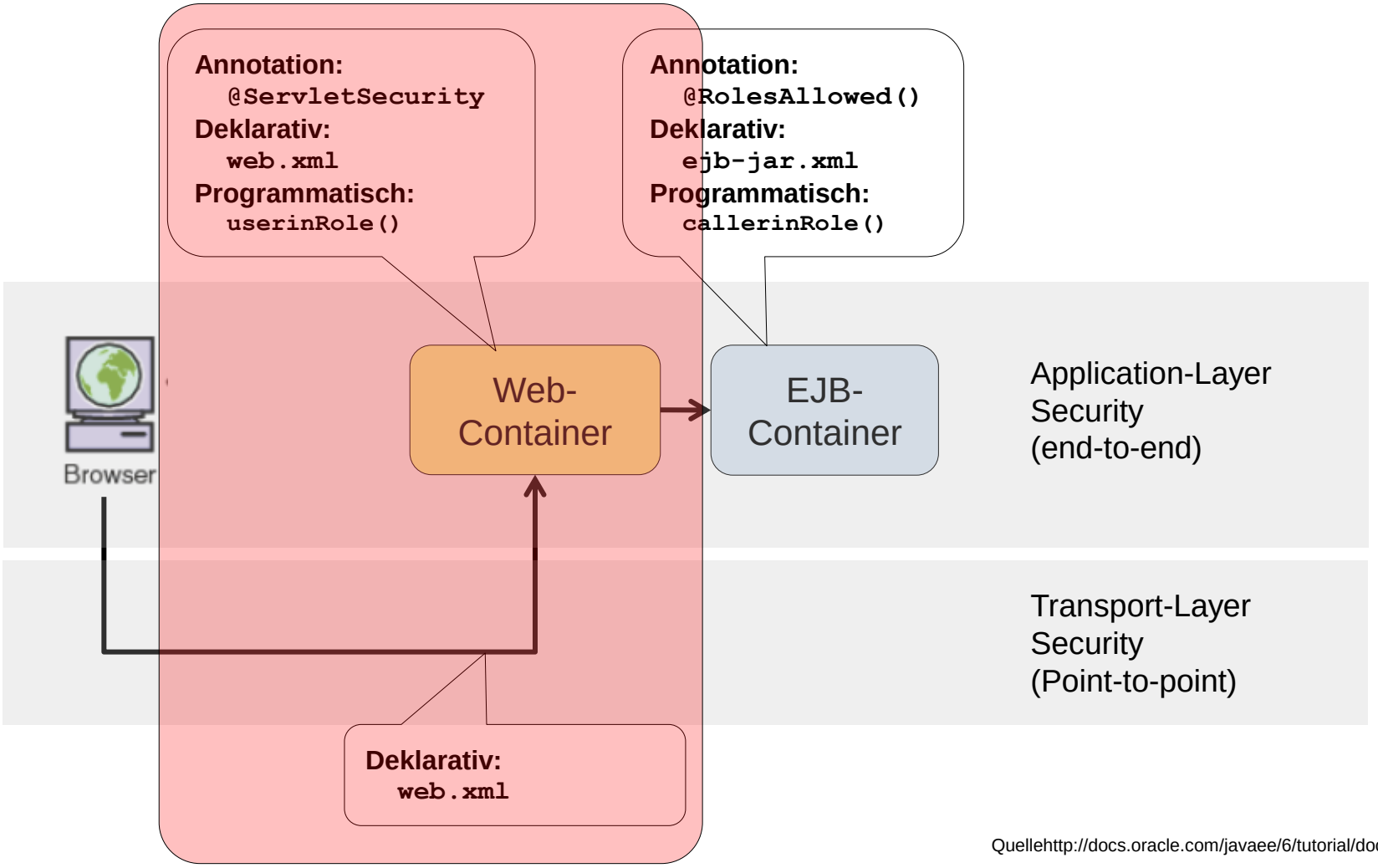
```
<glassfish-web-app>
    ...
    <security-role-mapping>
        <role-name>Admin</role-name>
        <principal-name>grabowski</principal-name>
    </security-role-mapping>

    <security-role-mapping>
        <role-name>Stud</role-name>
        <group-name>ai-stud</group-name>
        <group-name>winp-stud</group-name>
    </security-role-mapping>
    ...
</glassfish-web-app>
```

Web- oder EAR-Module:
`glassfish-web.xml`

EJB-Module
`glassfish-ejb-jar.xml`

oc/

Modul 15
Securing Web Applications



Quellehttp://docs.oracle.com/javaee/6/tutorial/doc/

Modul 15
Securing Web Applications

Task 1: Declaring Security Roles

Task 2: Specifying Authentication Mechanisms

Task 3: Specifying Security Constraints

Quellehttp://docs.oracle.com/javaee/6/tutorial/doc/

Task 1: Declaring Security Roles

```
 <!-- Security roles used by this web application -->
<security-role>
    <role-name>Admin</role-name>
</security-role>
<security-role>
    <role-name>Stud</role-name>
</security-role>
```

web.xml

Quellehttp://docs.oracle.com/javaee/6/tutorial/doc/

Task 2: Specifying Authentication Mechanisms

```
 <!-- Security roles used by this web application -->
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>file</realm-name>
    <form-login-config>
        <form-login-page>/login.xhtml</form-login-page>
        <form-error-page>/error.xhtml</form-error-page>
    </form-login-config>
</login-config>
```

web.xml

Quellehttp://docs.oracle.com/javaee/6/tutorial/doc/

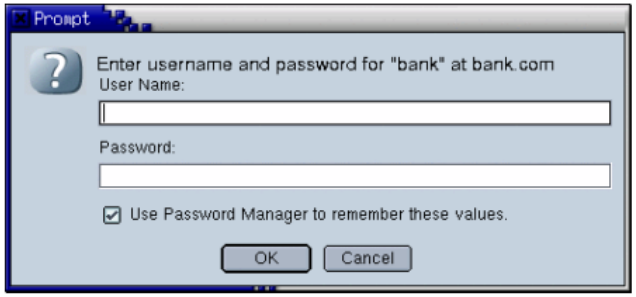
Modul 15
Securing Web Applications

`<auth-method>FORM</auth-method>`

Zu erstellen:
- **`login.xhtml`**
- **`error.xhtml`**

Bsp.: here

`<auth-method>BASIC</auth-method>`



Zu erstellen:
- -

Weitere Varianten: Digest authentication, Client authentication, Mutual authentication

Quellehttp://docs.oracle.com/javaee/6/tutorial/doc/

Task 3: Specifying Security Constraints

3.1 Specifying a Web Resource Collection
3.2 Specifying an Authorization Constraint
3.3 Specifying a Secure Connection

```
<security-constraint>
        <display-name>SecurityConstraint</display-name>

        <web-resource-collection>
            <web-resource-name>WRCollection</web-resource-name>
            <url-pattern>/studis/*</url-pattern>
        </web-resource-collection>

        <auth-constraint>
            <role-name>Stud</role-name>
        </auth-constraint>

        <user-data-constraint>
             <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>

</security-constraint>
```

CONFIDENTIAL: SSL
NONE: -

web.xml

Quellehttp://docs.oracle.com/javaee/6/tutorial/doc/

Web-Tier Security API

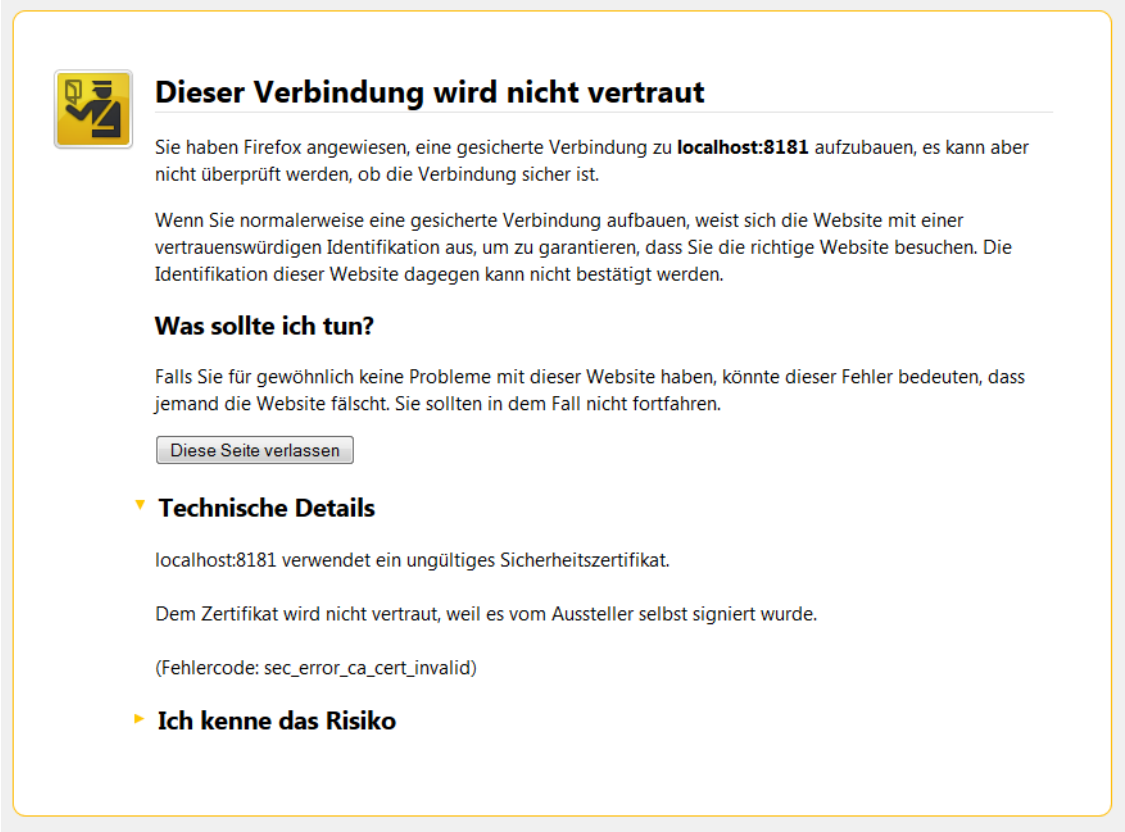
`getUserPrincipal()` **`isUserInRole()`**

```
String user = "nobody";
Principal p = request.getUserPrincipal();
if (p != null ) user = p.getName();
String message = "Welcome to HS, " + user + "<br>";
out.println(message);

if (request.isUserInRole("Stud")) {
    out.println("Student!");
} else {
    out.println("Looser!");
}
```

M15_ServletSec01

Quellehttp://docs.oracle.com/javaee/6/tutorial/doc/

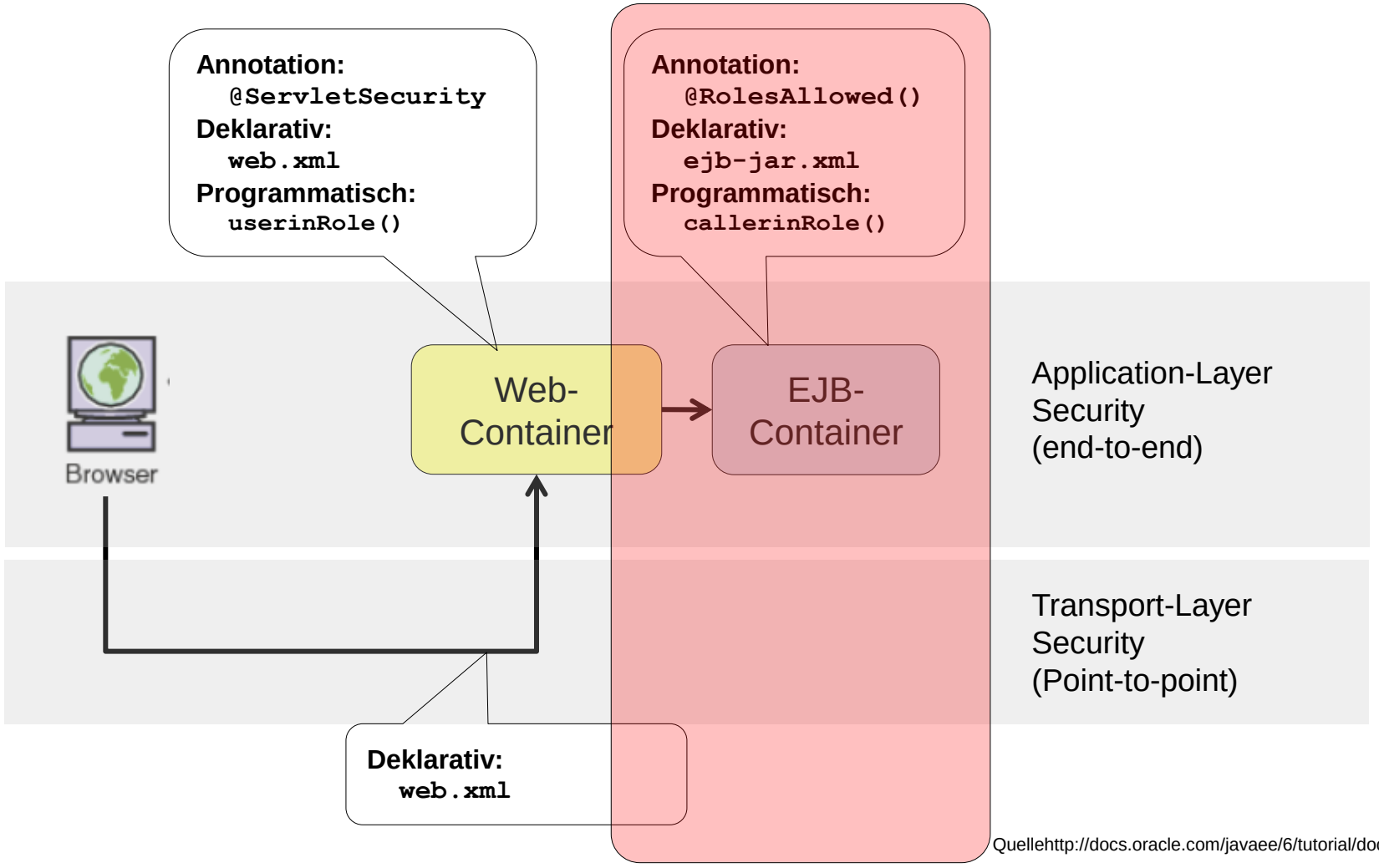
Modul 15
Using Programmatic Security with Web Applications



Kein Fehler,
Sondern fehlendes Zertifikat einer **„Certificate Authority”:**
Test free certificate

Quellehttp://docs.oracle.com/javaee/6/tutorial/doc/

Modul 15
Securing Enterprise Beans



Quellehttp://docs.oracle.com/javaee/6/tutorial/doc/

Modul 15
Declaring Security Roles

`@DeclareRoles`: Specifies all the roles that the application will use (…).

```
@DeclareRoles({"Administrator", "Manager", "Employee"})
```

`@RolesAllowed("list-of-roles"):` Specifies the security roles permitted to access methods in an application.

```
@DeclareRoles({"Administrator", "Manager", "Employee"})
public class Calculator {

    @RolesAllowed("Administrator")
    public void setNewRate(int rate) {
        ...
    }
}
```

`@PermitAll` (self-evident)

`@DenyAll` (self-evident)

Quellehttp://docs.oracle.com/javaee/6/tutorial/doc/

Modul 15
Securing an Enterprise Bean Programmatically

EJP-Tier Security API

`getCallerPrincipal()` **`isCallerInRole()`**

```
import java.security.Principal;
import javax.annotation.Resource;
import javax.annotation.security.DeclareRoles;
import javax.annotation.security.RolesAllowed;
import javax.ejb.*;

@Stateless @Remote @DeclareRoles(value={"Admin","Stud"})
public class HelloBean implements HelloService {

    @Resource SessionContext ctx;

    @RolesAllowed(value={"Admin","Stud"})
    public void getName(){
         String name = ctx.getCallerPrincipal().getName();
         System.out.println("Hello " + name);
         System.out.println("Your student status is:"
                 +  ctx.isCallerInRole("Stud"));
    }
}
```

Modul 15
Appendix: Secure Web-Services

Modul 15
Secure Web-Service

1. Define **Realms, Users, Groups**, and **Roles** for the (Glassfish-)Web-Server

2a) If **servlet based**-endpoints are used for JAX-WS or JAX-RS:

- Modify **`web.xml`** for
  (1) Declaring Security Roles,
  (2) Specifying Authentication Mechanisms and
  (3) Specifying Security Constraints

2b) If **EJB based**-endpoints are used for JAX-WS or JAX-RS:

- Modify **`glassfish-ejb-jar.xml`** (old: **`sun-ejb-jar.xml`**) for
  (1) Declaring Security Roles,
  (2) Specifying Authentication Mechanisms and
  (3) Specifying Security Constraints

A JAX-WS client can be restricted from accessing a web service if HTTP Authentication is enabled. The JAX-WS API can supply a user name and password if needed.

```
SayHelloService service = new SayHelloService();
SayHello port = service.getSayHelloPort();

((BindingProvider)port).getRequestContext().
                  put(BindingProvider.USERNAME_PROPERTY, "username");
((BindingProvider)port).getRequestContext().
                  put(BindingProvider.PASSWORD_PROPERTY, "password");
```

Access to the URL of the WSDL should not be restricted unless a local WSDL is provided to clients.

(Quick-trick: Leave HTTP-GET unprotected for *wscompile* and secure HTTP_POST only.)

A complete example for JAX-WS servlet endpoint and JAX-WS EJB endpoint see:
SSL and HTTP BASIC authentication with Glassfish and JAX-WS

Quelle: Developing Applications for the Java™ EE Platform , FJ-310-EE5, oracle

Modul 15
HTTP Security and JAX-RS clients

```
public class RestClient {

   public CustomerBean getCustomer() {
      Client client = new Client();
      client.addFilter(new HTTPBasicAuthFilter("USERNAME", "PASSWORD"));

      MultivaluedMap<String, String> queryParams = new MultivaluedMapImpl();
      queryParams.add("queryTerm", "someterm");

      WebResource webResource = client.resource("http://localhost/customer.xml");
      return webResource.queryParams(queryParams).get(CustomerBean.class);
   }
}
```

A complete example for secure JAX-RS endpoint see:
http://www.hascode.com/2010/11/creating-a-rest-client-step-by-step-using-jax-rs-jax-b-and-jersey/

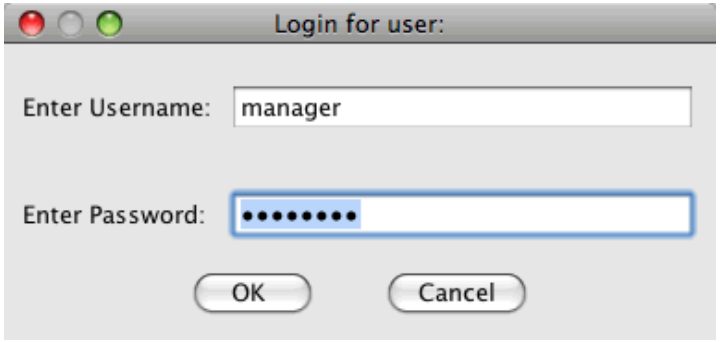
Modul 15
Building Secure Enterprise Beans in Java EE

1. Define **Realms, Users, Groups**, and **Roles** for the (Glassfish-)Web-Server

2. Annotate with **`@RolesAllowed`**

3. Implement the EJB-Client as usual

4. Running the application will ask for user & password



Full example at: https://netbeans.org/kb/docs/javaee/secure-ejb.html
or: Securing Remote EJB on Glassfish 2