JEE SECURITY IN JSF

Chapter 01 - Security

PROBLEM BEREICHE

Die Sicherung einer Applikation kann in unterschiedliche Aufgaben unterteilt werden

1. Konfiguration Applikation-Server
   - Configuring Security Realm
   - Setting up security repository (file or jdbc)
2. Authentication (method, login page, realm)
3. Konfiguration der Applikation Security
   - Setting up Groups and Roles in web.xml / glassfish-web.xml
   - Configuring Security for Application in web.xml
   - Enable secure communication (HTTPS) in web.xml
4. declarative und/oder programmatische Implementation der Security

1. KONFIGURATION APPLIKATION-SERVER

- Create a New Realm
  - Name: DemoRealm
  - className: com.sun.enterprise.security.auth.realm.file.FileRealm
  - JAAS Context: fileRealm
  - keyFile: ${com.sun.aas.instanceRoot}/config/DemoRealm.txt
- Manage User
  - userID: *nick* Group: *staff,admin* PW: *nick*
  - userID: *vicky* Group: *staff* PW: *vicky*

2. AUTHENTICATION

- Für die Authentifizierung der User verwenden wir "Form Based Authentication"
- Sobald ein User einen geschützten Bereich aufruft wird er zu einer Login-Seite weitergeleitet
- Schlägt das Login fehl, wird er auf eine entsprechende Seite weitergeleitet
- Der realm name muss mit dem realm name des Applikation-Server korrespondieren

2. AUTHENTICATION

- Auszug aus dem web.xml

Art der Authentifizierung

realm Name des Applikation-Server

Login Seite

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>DemoRealm</realm-name>
    <form-login-config>
        <form-login-page>/login/login.xhtml</form-login-page>
        <form-error-page>/login/login.xhtml</form-error-page>
    </form-login-config>
</login-config>
```

Seite bei fehlerhaftem Login

2. AUTHENTICATION

- Auszug aus login.xhtml
- JEE Standardnamen verwenden! Container handelt alles weitere

sendet das Formular als j_security_check (JEE Standard)

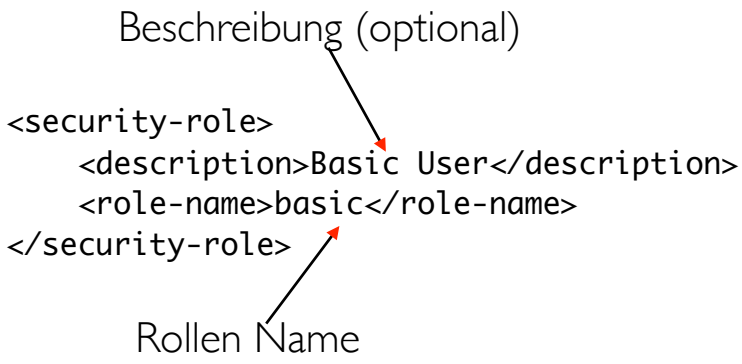
```
<h:form id="formLogin" prependId="false"
      onsubmit="document.getElementById('formLogin').action='j_security_check';">
      <h:panelGrid columns="2">

          <h:outputLabel for="j_username" value="Username: " />
          <h:inputText id="j_username" required="true" />

          <h:outputLabel for="j_password" value="Password: " />
          <h:inputSecret id="j_password" required="true" />

          <h:commandButton value="Login" type="submit" />

      </h:panelGrid>
  </h:form>
```

JEE Standard für Benutzername

JEE Standard für Passwort

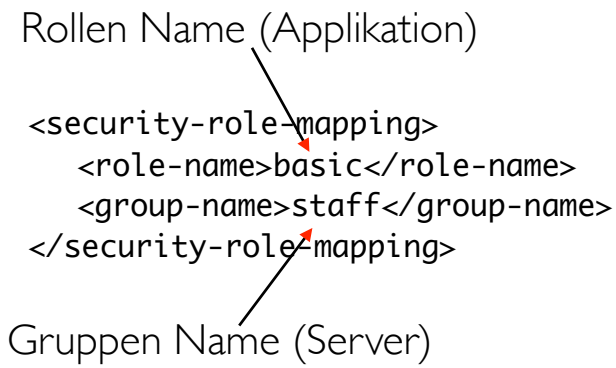
3. KONFIGURATION DER APPLIKATION SECURITY

- Setting up Roles in web.xml
- Die Roles werden später in den Constrains verwendet



3. KONFIGURATION DER APPLIKATION SECURITY

- Mapping Groups and Roles glassfish-web.xml
- Sollte *Default Principal To Role Mapping* im Applikation-Server nicht eingestellt sein oder andere mappings erwünscht



3. KONFIGURATION DER APPLIKATION SECURITY

- Configuring Security for Application in web.xml

Definition des Bereiches mittels URL

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Basic Content</web-resource-name>
        <url-pattern>/basic/*</url-pattern>
    </web-resource-collection>

    <auth-constraint>
        <description>only for authenticated users</description>
        <role-name>basic</role-name>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>
```

Definition der Rolle(n) mit Zugriff

3. KONFIGURATION DER APPLIKATION SECURITY

- Enable secure communication (HTTPS) in web.xml
- Kann auch in Zusammenhang mit auth-constraint (siehe vorherige Folie) definiert werden

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>…</web-resource-name>
        <url-pattern>…</url-pattern>
    </web-resource-collection>

    <user-data-constraint>
        <description>Secured Communication</description>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Definition des Bereiches mittels URL

Sichere Kommunikation erzwingen (HTTPS)

4. DECLARATIVE UND/ODER PROGRAMMATISCHE IMPLEMENTATION DER SECURITY

- Navigation zu geschützten Bereichen sollen lediglich Usern der entsprechenden Rollen zur Verfügung stehen
- Schützt nicht vor direktem Zugriff

```
<h:link value="go to Admin Page" outcome="/admin/admin"
   rendered="#{request.isUserInRole('admin')}" />
```

Prüfung auf Rolle

SESSION TIMEOUT

- Zur Erhöhung der Sicherheit sollte ein Session Timeout definiert werden (web.xml)

```
<session-config>
    <session-timeout>
        15
    </session-timeout>
</session-config>
```

Zeit in Minuten

FEHLER-SEITEN

- Benutzerdefinierte Fehlerseiten sollten zur besseren Benutzerführung implementiert werden (web.xml)
- Fehlerseiten können auf Exceptions oder auf HTTP Statuscodes definiert werden

Exception (z.B. Abgelaufene Session)

```
<error-page>
   <exception-type>javax.faces.application.ViewExpiredException</exception-type>
   <location>/status/error.xhtml</location>
</error-page>
```

HTTP Status Forbidden

```
<error-page>
    <error-code>403</error-code>
    <location>/status/forbidden.xhtml</location>
</error-page>
```

benutzerdefinierte Fehlerseite

LOGOUT

- Die Applikation sollte ein logout Methode zur Verfügung stellen welche die Session beendet (mittels ManagedBean)
- Die Applikation sollte den User auf eine Andere Seite weiterleiten (z.B. Login) oder die Seite neu laden
- Browser Cache (BackButton) muss mittels Filter unterdrückt werden

Invalidiert die Session

```
public String logout() {
   FacesContext.getCurrentInstance().getExternalContext().invalidateSession();
   return "/login/login.xhtml?faces-redirect=true";
}
```

erzwingt ein Redirect

Security Exercise 01: web security

- importiere das Projekt websecurity01
- 1. Set-Up Application-Server
  - Erstelle eine neues Security Realm im GlassFish Server (gemäss Folien)
  - Erstelle die User inklusive Gruppen (gemäss Folien)
- 2. Set-Up Authentication
  - Definiere die login-config (web.xml)
  - Implementiere eine Login-Seite (/login/login.xhtml)
- 3. Set-Up Application-Security
  - Definiere die security-roles (web.xml) und das Mapping (glassfish-web.xml)
  - Definiere Security-Constraints
    - für den Bereich /login/*
      - HTTPS Kommunikation
    - für den Bereich /basic/*
      - Nur für Authentifizierte Benutzer (Rolle: basic & admin)
    - für den Bereich /admin/*
      - Nur für admin User (Rolle: admin)
      - HTTPS Kommunikation
- Implementiere je eine Seite für die Bereiche basic (basic.xhtml) und admin (admin.xhtml)
- 4. In der Basic-Seite soll ein Link zur admin-Seite angezeigt werden, allerdings nur wenn der user zu den admin's gehört (Rolle: admin) Erstelle Fehler-Seiten für folgende errors
  - Session Timeout
  - HTTP Forbidden
- Implementiere ein ManagedBean (AdminBean) mit einer logout Methode welche den User ausloggt und Ihn auf die Login-Seite weiterleitet

JEE SECURITY

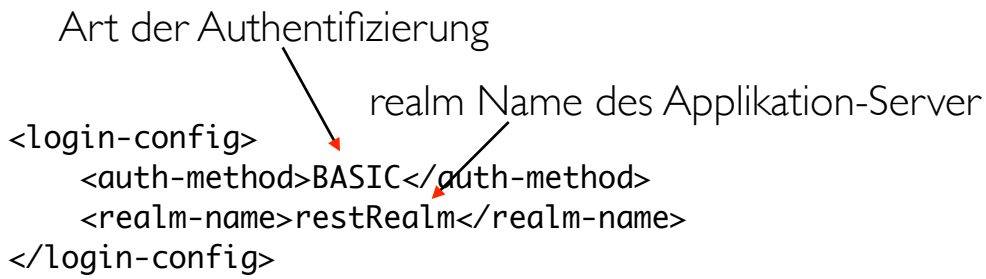
Chapter 02 - REST

2. AUTHENTICATION

- Für die Authentifizierung der User verwenden wir "Basic Authentication"
- Der Endpoint wird mittels URL-Pattern geschützt
- Der realm name muss mit dem realm name des Applikation-Server korrespondieren

2. AUTHENTICATION

- Auszug aus dem web.xml




```
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>restRealm</realm-name>
</login-config>
```

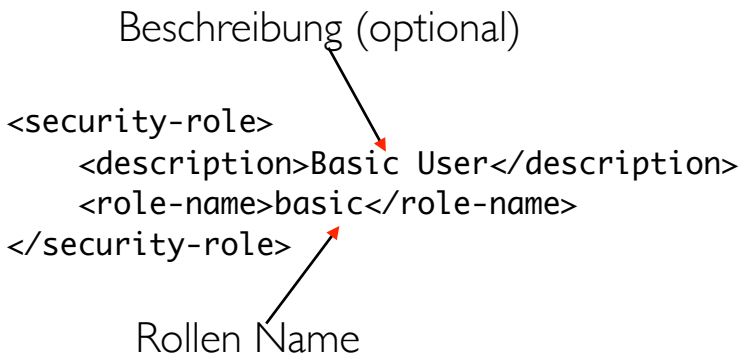

2. AUTHENTICATION

- Authorization header muss bei jedem Request gesendet werden
- Base 64-encoded string bestehend aus username, einem : und dem Passwort
- z.B:
  - String: dominik:geheim
  - base64: ZG9taW5pazpnZWhlaW0=

```
GET /websecurity02/rest/demo/admintest HTTP/1.1
Host: localhost:8888
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.3.1 (java 1.5)
Authorization: Basic ZG9taW5pazpnZWhlaW0=
```

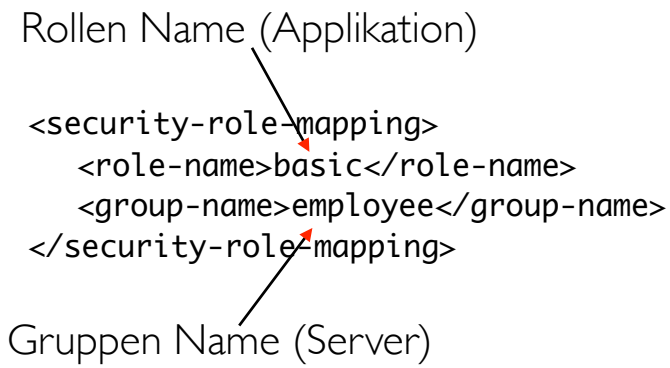
3. KONFIGURATION DER APPLIKATION SECURITY

- Setting up Roles in web.xml
- Die Roles werden später in den Constrains verwendet



3. KONFIGURATION DER APPLIKATION SECURITY

- Mapping Groups and Roles glassfish-web.xml
- Sollte *Default Principal To Role Mapping* im Applikation-Server nicht eingestellt sein oder andere mappings erwünscht



3. KONFIGURATION DER APPLIKATION SECURITY

- Configuring Security for Application in web.xml

Definition des Bereiches mittels URL

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>REST Content</web-resource-name>
        <url-pattern>/rest/*</url-pattern>
    </web-resource-collection>

    <auth-constraint>
        <description>only for authenticated users</description>
        <role-name>basic</role-name>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>
```

Definition der Rolle(n) mit Zugriff

3. KONFIGURATION DER APPLIKATION SECURITY

- Enable secure communication (HTTPS) in web.xml
- Sehr wichtig, da Username & Passwort bei jedem Request mitgesendet werden

Definition des Bereiches mittels URL

```
<user-data-constraint>
    <description>Secured Communication</description>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
```

Sichere Kommunikation erzwingen (HTTPS)

3. KONFIGURATION DER APPLIKATION SECURITY

```
<security-constraint>

    <web-resource-collection>
        <web-resource-name>rest content</web-resource-name>
        <url-pattern>/rest/*</url-pattern>
    </web-resource-collection>

    <auth-constraint>
        <description>following roles can access this resource</description>
        <role-name>admin</role-name>
        <role-name>basic</role-name>
    </auth-constraint>

    <user-data-constraint>
        <description>Secured Communication</description>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>

</security-constraint>
```


4. DECLARATIVE UND/ODER PROGRAMMATISCHE IMPLEMENTATION DER SECURITY

- Zusätzlich kann nun auch declarative und/oder programmatische Sicherheit implementiert werden

- Deklarative
  - @RolesAllowed(“admin“)
- Programmatisch
  - isUser inRole(“admin“)

4. PROGRAMMATISCHE IMPLEMENTATION DER SECURITY

- Programmatische Sicherheit kann in JAX-RS mittels SecurityContext implementiert werden
- public interface SecurityContext
  - public Principal getUserPrincipal();
  - public boolean isUserInRole(String role);
  - public boolean isSecure();
  - public String getAuthenticationScheme();
- Der Context wird mittels Injection direkt der REST Methode bereitgestellt
  - public Response test(@Context SecurityContext sec){}

5. ROLLING YOUR OWN APPROACH

- Authorization kann natürlich auch mittels Header (oder anderen) Parameter mitgegeben werden
- Login muss anschliessend in den einzelnen REST Methoden durchgeführt werden
- Deklarative sowie URL Pattern Security kann nicht ohne weiteres realisiert werden
- Zu diesem Zweck wird der HttpServletRequest Context Injectet

```
@Context
private HttpServletRequest request;
…

request.login(username, password);
```