Seminararbeit im Fach: Aktuelle Technologien zur Entwicklung verteilter Java-Anwendungen

Thema:
Sicherheit in unternehmenskritischen Applikationen

Vorgelegt von:	**Christoph Florian Marketsmüller**
Matrikelnummer:	**05733413**
Semester:	**4.Semester Wirtschaftsinformatik**
Gruppe:	**IB4B**
Aufgabensteller:	**Michael Theis**
Abgabedatum:	**08.05.2015**

Gliederung

1. Einführung in Java EE7 Security ... 3

Grundlagen ... 3

2. Wichtige Begriffe der Java-Security ... 4

3. Arten der Sicherheiten ... 5

2.1 Declarative security ... 5

Der Deployment Descriptor ... 5

Annotations ... 5

2.2 Programmatic security ... 5

4. Sicherheits Mechanismen ... 6

3.1 Java SE Mechanismen ... 6

3.2 Java EE Mechanismen ... 6

3.2.1 Application-Layer Security ... 6

3.2.2 Transport-Layer Security ... 7

3.2.3 Messages-Layer Security ... 7

5. Sicherheit von Web Anwendung ... 7

5.1 Deklarative Sicherheit ... 7

5.1.1 Security Constraints ... 7

5.1.2 Authentisierungsmechanismen ... 10

5.1.3 Bestimmung von Sicherheitsrollen ... 13

5.2 Programmatische Sicherheit ... 14

5.2.1 HttpServlet ... 14

5.2.2 HttpServletRequest ... 15

5.2.3 HttpServletResponse ... 17

6. Sicherheit auf Enterprise JavaBeans Ebene(EJB) ... 18

6.1 Deklarative Sicherheit ... 18

6.1.1 Wichtige Annotations ... 18

6.1.2 Die ejb-jar.xml Datei ... 19

6.2 Programmatisch Sicherheit ... 20

7. Fazit ... 21

8. Literaturverzeichnis ... 21

1. Einführung in Java EE7 Security

In dieser Arbeit werden die Sicherheitsspezifikationen von der Java Enterprise Edition (JEE) vorgestellt.

Da fast jedes Unternehmen Ressourcen über ein öffentliches Netzwerk bereitstellt, aber nicht alle Ressourcen für jeden sichtbar sein sollen, bietet Java EE einige Sicherheitsmechanismen um die Zugriffe auf die verschiedenen Ressourcen zu regeln.

Ziel der Arbeit ist es die verschieden Sicherheitsmechanismen von Java EE vorzustellen, und zu zeigen an welchen Stellen einer Java EE Anwendung diese zur Geltung kommen. Und um klarzustellen welche Mittel für welche Situation am besten zu verwenden sind.

Der Aufbau der Arbeit ist wie folgt:

- Grundlagen / Vorteile von Java EE7 Security
- Welche Arten von Sicherheit Mechanismen gibt es eigentlich?
- Sicherheit einer Web Anwendung
- Sicherheit auf EJB Ebene

Grundlagen[1]

Die Einbringung der Java EE7 Security Mechanismen bringt verschiedene Funktionen mit sich.
Es werden die Zugriffe auf die verschiedenen Ressourcen verwaltet, die Benutzer können für ihre Tätigkeiten belangt werden und das System wird von Service Unterbrechungen geschützt, die die Qualität der Anwendung beeinflusst.

Außerdem bietet es folgende Vorteile:

- Vereinfachung der Administration
- Die Anwendung werden transparenter

Java EE7 Security Mechanismen bringen unter anderen folgende Eigenschaften mit sich:

- Daten Integrität: Die Daten Integrität sorgt dafür, dass die Informationen, die gerade von einen Benutzer bearbeitet werden nicht von einem Dritten benutzt werden können.
- Nachvollziehbarkeit: Es ist nachzuvollziehen wer wann was geändert hat.[2]

[1] Oracle, 24.04.2015, http://docs.oracle.com/javaee/7/tutorial/security-intro001.htm#BNBWK
[2] Oracle, 24.04.2015, https://docs.oracle.com/javaee/7/tutorial/security-intro001.htm

2. Wichtige Begriffe der Java-Security

In Java EE7 gibt es wichtige Begriffe im Bereich der Security, die öfters wieder auftauchen werden.

Authentisieren[3]

Die Authentisierung ist für die Identifikation des Users verantwortlich.

Autorisierung[3]

Die Autorisierung überprüft welche Rechte der angemeldet Benutzer besitzt.

Realms[4]

Die geschützten Ressourcen auf einen Server können separat über mehrere Realms geschützt werden. In einem Realm wird festgelegt welche Benutzer welcher Gruppe zugehören und welche Rollen sie einnehmen dürfen.

Bei einen Glassfish Server sind zum Beispiel der file, der admin-realm und der certifcate Realm vorgegeben.

Dabei kümmert sich der File Realm um das lokale Abspeichern der Benutzerzugangsdaten und ist somit für die Authentifikation zwischen Client und Server verantwortlich.

Der admin-realm ist wie der File Realm für Benutzerzugangsdaten zuständig, nur werden hier die Administratoren verwaltet.

Bei dem certificate Realm werden die User in einer Zertifizierten Datenbank abgelegt. Hierbei nutzt der Server Zertifikate um die Verbindung mit den Web-Clients herzustellen. Und die Verifizierung der Benutzer geschieht über eine X.509 Zertifikat.

User[5]

Ein Benutzer(User) ist eine Identität, die in einem Realm definiert ist. Mit dieser Identität wird entweder eine reelle Person oder ein Anwendungsprogramm identifiziert.

Groups[6]

Gruppen(Groups) sind eine Zusammenfassung von mehreren Benutzern mit gleichen Eigenschaften.

Roles[7]

Die Rollen(Roles) legen fest welche Zugriffsberechtigungen ein Benutzer oder eine Gruppe hat.

Principal[8]

Ein Principal ist eine Entität(ein bestimmtes Objekt), die über ein Authentifizierungs protokoll identifiziert wird. Ein Principal kann zum Beispiel eine Person, ein Name, ein Unternehmen, oder eine Log in ID sein.

[3] Oracle, 24.04.2015, Kapitel 47.1.3 http://docs.oracle.com/javaee/7/tutorial/security-intro001.htm#BNBWK
[4] Oracle, 24.04.2015, Kapitel 47.5.1.1 http://docs.oracle.com/javaee/7/tutorial/security-intro005.htm#BNBXJ
[5] Oracle, 24.04.2015, Kapitel 47.5.1.2 http://docs.oracle.com/javaee/7/tutorial/security-intro001.htm#BNBWK
[6] Oracle, 24.04.2015, Kapitel 47.5.1.3 http://docs.oracle.com/javaee/7/tutorial/security-intro001.htm#BNBWK
[7] Oracle, 24.04.2015, Kapitel 47.5.1.4 http://docs.oracle.com/javaee/7/tutorial/security-intro001.htm#BNBWK
[8] Oracle, 24.04.2015, Kapitel 47.5.1.5 http://docs.oracle.com/javaee/7/tutorial/security-intro001.htm#BNBWK

Subject[8]
Ein Subject ist eine Sammlung von Informationen zu einer bestimmten Entität.

Session[9]
Bei einer Session werden die Benutzer Daten in einer „ID“ gespeichert, die der Webserver dem Client mitteilt. Durch diese „ID“, die bei allen weiteren Anfragen des Clients mitgeschickt wird, kann der Webserver den Client authentisieren. Dies bringt den Vorteil mit sich, dass sich ein Benutzer nicht für jede Seite neu anmelden muss.

3. Arten der Sicherheiten[10]

Es gibt zwei verschieden Arten von Sicherheitsmechanismen, die Java EE bietet. Diese werden von sogenannten Securing Containern bereitgestellt.

Beide Arten können Unabhängig voneinander verwendet werden.

3.1 Declarative security
Der erste Mechanismus ist die deklarative Sicherheit (Declarative security). Diese beschreibt welchen Anforderungen eine Anwendung erfüllen muss

Diese kann durch eine „Einsatzbeschreibung“ (Deployment Descriptor) oder durch Annotations eingesetzt werden.

Die Annotations und der Deployment Descriptor (eine externe XML Datei) beschreiben den strukturellen Aufbau einer Anwendung.

Die Declarative security wird vor allem für die Authentisierung und die Autorisierung in der Anwendung benutz.

Der Deployment Descriptor
Im Deployment Descriptor ist eine externe XML Datei in der die verschiedenen Sicherheitsregeln festgelegt werden.

Annotations
Das Festlegen der Sicherheitsrollen kann auch durch des verwenden von Annotations erfolgen.

Diese werden in den verschiedenen Servlets oder Klassen, auf den verschiedenen Ebenen genutzt.

3.2 Programmatic security[11]
Der zweite Mechanismus ist die programmatische Sicherheit die direkt in das System eingebettet wird. Sie wird benutz, wenn die Erklärende Sicherheit alleine nicht mehr ausreicht. Sie wird benutzt um logische Entscheidungen zu treffen, die anhand der aktuellen Rolle bestimmt wird.

[9] Luke404, 04.05.2015, Kommentar am 27. September 2010 um 13:44, http://stackoverflow.com/a/3804387
[10] Oracle, 04.05.2015, Kapitel 47.1, http://docs.oracle.com/javaee/7/tutorial/security-intro001.htm#BNBWK
[11] Oracle, 04.05.2015, Kapitel 47.3.3 http://docs.oracle.com/javaee/7/tutorial/security-intro003.htm#BNBXH

4. Sicherheits Mechanismen[12]

Java EE stellt einige Mechanismen zur Verfügung, die für Anwendungssicherheit zuständig sind. Diese können alle individuell auf die eigene Anwendung spezifiziert werden. Die Verknüpfung dieser Mechanismen ist möglich und meist sinnvoll, da sich jeder einzelne Mechanismus auf seinen Service spezialisiert.

4.1 Java SE Mechanismen[13]

Java SE stellt folgende Mechanismen für die Sicherheit zur Verfügung:

- Java Authentication and Authorization Service(JAAS)
- Java Generic Security Services(Java GSS-API)
- Java Cryptography Extension(JCE)
- Java Secure Sockets Extension(JSSE)
- Simple Authentication and Security Layer(SASL)

4.2 Java EE Mechanismen[14]

Java EE stellt 3 Mechanismen für die Sicherheit zur Verfügung. Die einzelnen Mechanismen werden nachfolgend näher behandelt.

- Application-Layer Security
- Transport-Layer Security
- Message-Layer Security

4.2.1 Application-Layer Security[15]

Die Application-Layer Security kümmert sich um einen Sicheren Kommunikations Stream der Anwendung und der Schutz der Ressourcen von Angriffen von außen.

Es bringt die Vorteile mit sich, dass sich die Sicherheitsmaßnahmen leicht und genau auf die Anwendung einstellen lassen, und das sehr spezifiziert.

Was wiederrum den Nachteil mit sich bringt, dass die Sicherheitsmaßnahmen nur auf die eine Anwendung zutreffen und nicht auf andere Anwendungen übertragbar sind. Ein weiterer Nachteil besteht darin, wenn an der Anwendung zum Beispiel ein Web-Service angefügt werden soll, das der Application-Layer nur Sicherheit über die Daten der Anwendung bieten kann und nicht über die Daten die oft über verschiedene Wege übertragen werden.

[12] Oracle, 04.05.2015, Kapitel 47.2 http://docs.oracle.com/javaee/7/tutorial/security-intro002.htm#BNBWY
[13] Oracle, 04.05.2015, Kapitel 47.2.1 http://docs.oracle.com/javaee/7/tutorial/security-intro002.htm#BNBWY
[14] Oracle, 04.05.2015, Kapitel 47.2.2 http://docs.oracle.com/javaee/7/tutorial/security-intro002.htm#BNBWY
[15] Oracle, 04.05.2015, Kapitel 47.2.2.1 http://docs.oracle.com/javaee/7/tutorial/security-intro002.htm#BNBWY

4.2.2 Transport-Layer Security[16]

Die Transport-Layer Security sorgt sich um eine sichere Datenübertragung zwischen den Clients und den Servern mittels Secure Socket Layer(SSL). Mit der sicheren Punkt zu Punkt Verbindung wird die Nachrichten Integrität und die Authentisierung sichergestellt. Die Bereitstellung dieser Verbindung passiert folgendermaßen.

- Die Seiten einigen sich auf eine geeignete Verschlüsselung zur Authentisierung.
- Während des Datenaustausches wird diese Verschlüsselung auf beiden Seiten gespeichert.

Ein Nachteil des Transport-Layer Security ist, dass eine Application-Layer Security vorausgesetzt wird. Ein weiterer Nachteil ist, dass nur die Sicherheit während des Transports gesichert ist und somit ihre Sicherheit bei Erreichen des Endpunktes verliert.

4.2.3 Messages-Layer Security[17]

Die Messages-Layer Security kümmert sich um das sicherere Versenden von Nachrichten bei der Ende zu Ende Beziehungen. Die Nachricht wird hier bei dem Sender so verschlüsselt, dass sie nur der wirkliche Empfänger entschlüsseln kann und nicht irgendwelche Zwischenknoten an denen die Nachricht eventuell vorbeigeschickt wird.

Die Vorteile von Messages-Layer Security sind unter anderem, die Sicherheit das die Nachricht beim Empfänger wirklich ankommt, die Versendung der Nachricht kann ohne Probleme über mehrere Zwischenknoten laufen und die Sicherheit ist Unabhängig von der Anwendungsumgebung und der Transport Protokolle.

Der Nachteil an Messages-Layer Security ist, dass das Einrichten aufwendiger und komplizierter ist.

5. Sicherheit von Web Anwendung

Um bestimmte Ressourcen zu sichern, werden auf der Ebene der Web Anwendung einige Sicherheitsregeln definiert, damit nicht jeder User auf jede Seite zugreifen kann. Dies kann wie oben geschildert(Kapitel 3) auf zwei verschieden Arten festgelegt werden. Die deklarative Sicherheit und die programmatische Sicherheit.

5.1 Deklarative Sicherheit

5.1.1 Security Constraints[18]

Um nicht für jede einzelne Ressource Sicherheitsregeln festzulegen werden sogenannte Security Constraints angelegt, in diesen werden mehrere Ressourcen zusammengefasst, für die dann die Sicherheitsregeln definiert werden.

[16] Oracle, 04.05.2015, Kapitel 47.2.2.2 http://docs.oracle.com/javaee/7/tutorial/security-intro002.htm#BNBWY
[17] Oracle, 04.05.2015, Kapitel 47.2.2.3 http://docs.oracle.com/javaee/7/tutorial/security-intro002.htm#BNBWY
[18] Oracle, 05.05.2015, Kapitel 48.2.1 http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#GKBAA

Deployment Descriptor

Falls die Anwendung kein Servlet nutzt, müssen die Constraints in einem Deployment Descriptor durch ein *security-constraint* Element festgelegt werden.

Das *security-constraint* Element kann folgende Subelemente beinhalten.

- Web rescource collection
- Authorization constraint
- User data constraint

Web resource collection[19]

In dem *web-rescource-collection* Element wird aus folgenden zwei unter Elementen gebildet.

Das erste Unterelement ist das *web-resource-name* Element. Dies ist nur optional und wird genutzt um verständlich zu machen, welcher Bereich an URLs jetzt aufgezählt wird(Zum Beispiel die Seiten die nur der Administrator sehen darf).

Das zweite Unterelement ist das *url-pattern* Element. Hier werden alle URLs aufgezählt, die geschützt werden sollen.

Es können auch die verschiedenen HTTP Methoden festgelegt werden, mit denen der Zugriff auf die geschützen Ressourcen erlaubt ist. Dies passiert mit dem *http-method* Element.

Authorization constraint[20]

Das *auth-constraint* Element wird in das *web-resource-collection* Element eingebettet. Es legt fest, welche Rollen auf die Seiten zugreifen dürfen. Diese Rollen müssen mit den Rollen übereinstimmen die man separat in dem *security-role* Element ablegt(Rollen sind case sensitive).

User data constraint[21]

Das User data constraint ist für eine Sichere Verbindung zuständig und besteht aus dem Element *transport-guarantee* das einen von drei zulässigen Parametern erwartet.

- CONFIDENTAL stellt sicher, dass die Nachricht erfolgreich gesendet wird und dass kein Dritter die Nachricht lesen kann. Die Einstellung legt fest, dass ein Aufruf der betroffenen Seiten mittels HTTP automatisch in HTTPS dargestellt wird(dafür wird ein SSL Connector benötigt).

- INTEGRAL: hingegen, stellt nur sicher, ob die Nachricht erfolgreich gesendet wird.

- NONE: bedeutet, dass der Container die Nachricht annehmen muss. Egal ob die Verbindung gesichert ist oder nicht.

[19] Oracle, 04.05.2015, Kapitel 48.2.1.1, http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#GKBAA
[20] Oracle, 04.05.2015, Kapitel 48.2.1.2, http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#GKBAA
[21] Oracle, 04.05.2015, Kapitel 48.2.1.3, http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#GKBAA

Erläuterung Anhand eines Beispiels[22]: (Deployment Descriptor security constraints)

```
<security-constraint>

<!-- Festlegung der Ressourcen die geschützt werden sollen -->

    <web-resource-collection>

        <web-resource-name>wholesale</web-resource-name>

<!-- Konkrete Seite die geschützt werden soll -->

        <url-pattern>/acme/wholesale/*</url-pattern>

    </web-resource-collection>

    <auth-constraint>

<!-- Festlegung welche Rollen auf die Seiten Zugriff haben -->

        <role-name>PARTNER</role-name>

    </auth-constraint>

    <user-data-constraint>

<!-- Festlegung der Sicherheitsstufe der Verbindung -->

        <transport-guarantee>CONFIDENTIAL</transport-guarantee>

    </user-data-constraint>

</security-constraint>
```

Bemerkung: In der Praxis werden auf Java EE Servern CONFIDENTAL und INTEGRAL gleich behandelt.

Annotation[23]

Wenn die Anwendung ein Servlet nutzt, können die Constraints mit Annotations festgelegt werden. Dies ist zu empfehlen, da hier die Constraints direkt am Servlet gehängt werden und die Informationen nicht in eine externe Datei ausgelagert werden müssen. Hierzu werden die @ServletSecurity Annotation mit Verbindung zu der @HttpConstraint oder den@HttpMethodConstraint Annotation verwendet.

[22] Oracle, 04.05.2015, Kapitel 48.2.1.4 http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#BNCBN

[23] CodeJava, 05.05.2015, http://www.codejava.net/java-ee/servlet/servletsecurity-httpmethodcontraint-and-httpconstraint-annotations-examples

Erläuterung anhand mehrerer Beispiele[24]: (Annotation_security constraints)

```
@WebServlet("/manage")
@ServletSecurity(
 httpMethodConstraints = {

      // Hier wird festgelegt das der Benutzer für HTTP POST und GET
      // Methoden in der Rolle „admin" sein muss. Für die POST Methode
      ist
      // eine Sichere Verbindung notwendig. Alle anderen HTTP Methoden
      // haben keine Einschränkungen.

  @HttpMethodConstraint(value = "GET", rolesAllowed = "admin"),
  @HttpMethodConstraint(value = "POST", rolesAllowed = "admin",
                        transportGuarantee =
TransportGuarantee.CONFIDENTIAL),
 }
)
public class AdminServlet extends HttpServlet {
    // servlet code...
```

5.1.2 Authentisierungsmechanismen[25]

Die Authentisierungsmechanismen legt fest, wie umfangreich der Authorisierungsprozess gestaltet wird. Es müssen folgende Funktionen verwaltet werden:

- Wie kann ein User auf die Ressourcen zugreifen
- Wie wird der User standardmäßig authentifiziert
- Wie kann sich ein User authentifizieren

Um diese Authentisierungsmechanismen zu verwirklichen, wird allerdings erst eine Datenbank auf den Server benötigt, in der folgende Daten abgespeichert sind. Und es wird ein Deployment Descriptor benötigt in dem der Authentisierungsmechanismus festgelegt ist, da dies nicht durch Annotations geschehen kann[26]. Welcher Benutzer welches Passwort und welcher Rolle besitzt.

Dafür unterstützt Java EE 5 Authentisierungsmechanismen.

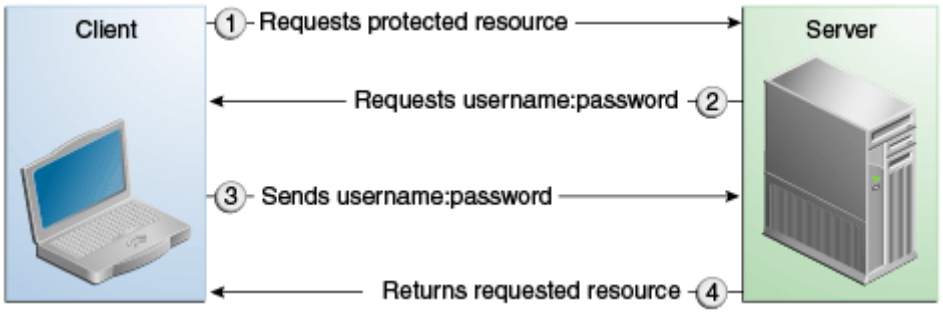
- Basis Authentisierung
- Form basierte Authentisierung
- Digest Authentisierung
- Client Authentisierung
- Beidseitige Authentisierung

[24] CodeJava, 05.05.2015, @ServletSecurity annotation exampels http://www.codejava.net/java-ee/servlet/servletsecurity-httpmethodcontraint-and-httpconstraint-annotations-examples
[25] Oracle, 05.05.2015, Kapitel 48.2.2 http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#BNCBN
[26] Oracle, 07.05. 2015, Überschrift: Specifying Security Constraints http://docs.oracle.com/javaee/6/tutorial/doc/gkbaa.html

Basis Authentisierung[27]

Die Basis Authentisierung ist die Authentisierung die standardmäßig vorgegeben wird, wenn keine spezielle Festlegung stattgefunden hat.

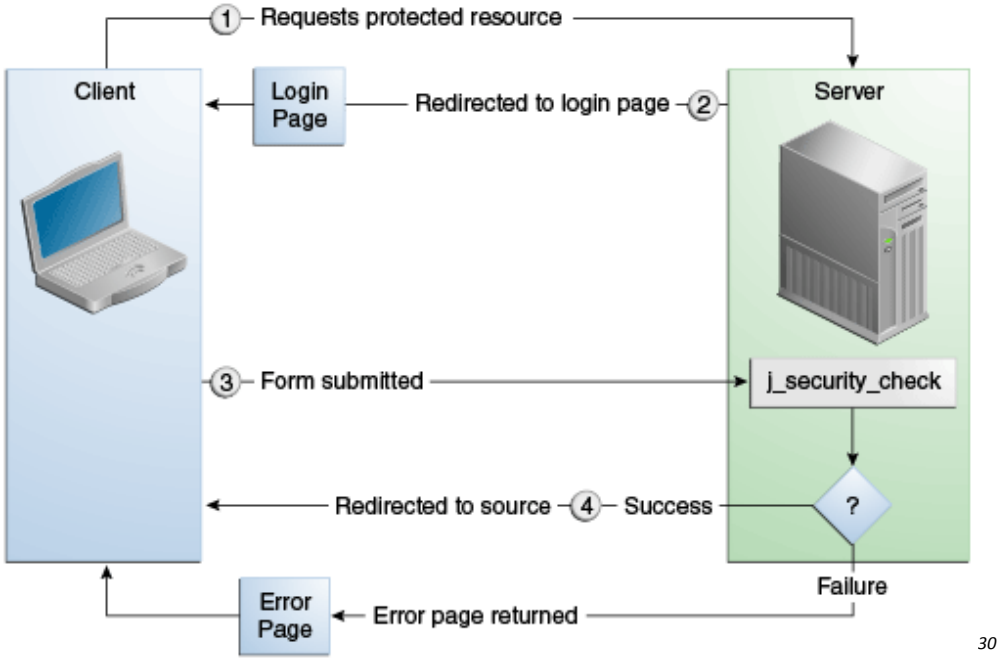
Wie im unteren Bild gezeigt wird, wird bei der Basis Authentisierung vom Client eine Anfrage an den Server auf beschütze Ressourcen geschickt. Daraufhin bittet der Server um die Anmelde Daten des Clients. Bekommt der Server vom Client die Daten des Users zugeschickt, vergleicht er diese mit seiner Datenbank. Sind die übergebenen Daten korrekt schickt der Server dem Client die geforderten beschützen Ressourcen. Falls die Authentisierung fehlschlägt wird der Client auf eine Standardfehlerseite zurückgewiesen(HTTP 401)[28].



29

Form basierte Authentisierung

Die Formbasierten Authentisierung ist ähnlich wie die Basis Authentisierung aufgebaut, nur dass der Entwickler bei der Formbasierten Authentisierung Freiraum bei der Gestaltung der Login Seite hat.



30

[27] Oracle, 05.05.2015, Kapitel 48.2.2.1 http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#BNCBN

[28] Oracle, 05.05.2015, Kapitel: Allow Retries http://docs.oracle.com/cd/E39820_01/doc.11121/gateway_docs/content/authn_http_basic.html

[29] Oracle, 05.05.2015, Kapitel 48.2.2.1 Figure 48-1 HTTP Basic Authentication http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#BNCBN

[30] Oracle, 05.05.2015, Kapitel 48.2.2.1 Figure 48-2 HTTP Form-Based Authentication http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#BNCBN

Diese beinhaltet das Form Element von HTTP in der ein User seine Daten eingibt und abschickt. Diese Daten werden meist über die HTTP POST Methode gesendet[31]. Sind die Daten dann mit der Serverdatenbank übereinstimmend, schickt der Server die gewünschten Ressourcen den Client, ansonsten wird eine Fehler Seite zurückgegeben die vom Entwickler gewünscht wird.

Digest Authentisierung[32]

Die Digest Authentisierung beruht auf der Basis Authentisierung mit Ausnahme, dass die Benutzerdaten vor dem Senden verschlüsselt werden, bevor sie über das Netzwerk geschickt werden

Die Entschlüsselung erfolgt dann beim Server, der die Daten dann wieder mit seiner Datenbank vergleicht.

Client Authentisierung

Bei der Client Authentisierung authentisiert der Server den Client durch ein Schlüssel Zertifikat[33]. Dieses Schlüssel Zertifikat ist sozusagen der Digitale Ausweis[34].

Beidseitige Authentisierung

Die beidseitige Authentisierung wird wegen zeittechnischen Gründen nicht behandelt. Genauere Informationen finden sie allerdings unter:

http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#BNCBN Kapitel 50.2.2.

Deployment Descriptor[35]

Um einen Authentisierungsmechanismus zu wählen, muss dieser im Deployment Descripotr in dem Element ***login-config*** definiert werden. Das Element login-config besteht aus folgenden Elementen:

- auth-method
- realm-name
- form-login-config

Auth-method

In dem ***auth-method*** Element wird ein Wert erwartet der entweder NONE, BASIC, DIGEST, FORM oder CLIENT-CERT ist. Dieser Wert legt fest welcher Authentisierungsmechanismus gewählt wird. Wird als Wert NONE gewählt ist die Authentisierung von dem Benutzer nicht nötig.

Realm-name

Danach kommt das ***realm-name*** Element, dass den Realm Namen bestimmt, falls die Authentisierung der Basis Authentisierungsmechanismus ist.

Form-login-config

Das ***form-login-config*** Element bestimmt welche Login Seite und welche Fehler Seite angezeigt werden soll.

Erläuterung Anhand eines Beispiels[36]: (Deployment Descirptor_Authentisierungsmechanismus)

---

[31] Oracle: 05.05.2015, Kapitel Overview http://docs.oracle.com/cd/E39820_01/doc.11121/gateway_docs/content/authn_html_form.html

[32] Oracle, 05.05.2015, Kapitel 48.2.2.3 http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#GKBAA

[33] Bsi Bund, 05.05.2015, Kapitel Darstellung PKI Allgemein https://www.bsi.bund.de/DE/Themen/ElektronischeAusweise/Sicherheitsmechanismen/sicherPKI/pki_node.html

[34] Oracle, 05.05.2015, Kapitel 50.2.1 http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#BNCBN

[35] Oracle, 05.05.2015, Kapitel 48.2.3 http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#BNCBN

```
<login-config>

<!-- Festlegung des Authentisierungsmechanismus -->

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

<!-- Festlegung des Realm Namen -->

    <realm-name>file</realm-name>

    <form-login-config>

<!-- Festlegen der Login Seite -->

        <form-login-page>/login.xhtml</form-login-page>

<!-- Festlegen der Fehler Seite -->

        <form-error-page>/error.xhtml</form-error-page>

    </form-login-config>

</login-config>
```

5.1.3 Bestimmung von Sicherheitsrollen

Die Rollen werden im Deployment Descriptor durch das Element *security-role* festgelegt. Das *security-role* Element beinhaltet das Element *role-name*, indem der tatsächliche Rollen Name abgespeichert wird.

[36] Oracle, 05.05.2015, Kapitel 48.2.3, http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#GKBAA

Erläuterung Anhand eines Beispiels:[37] *(Rollen)*

```
<!-- Security roles used by the web application from the exampel
(Deployment Descriptor security constraint -->

.

.

Code from example (Deployment Descriptor security constraint)

<security-role>

    <role-name>PARTNER</role-name>

</security-role>

<security-role>

    <role-name>NOPARTNER</role-name>

</security-role>
```

5.2 Programmatische Sicherheit[38]

Die Programmatische Sicherheit wird verwendet wenn die Deklarative Sicherheit nicht ausreichend für die Webanwendung ist. Hierfür wird das HttpServlet Interface genutzt.

5.2.1 HttpServlet[39]

Die Abstrakte Basis Klasse HttpServlet stellt für jede HTTP Methode ein Methode zur Verfügung.

Eine Klasse die von HttpServlet erbt, muss mindestens eine dieser Methoden überschreiben.(Quelle oracle dokumentation)

Die Methoden erwarten ein HttpServletRequest Objekt, das die Anfrage eines Webbrowser repräsentiert und ein HttpServletResponse Objekt, das die Antwort des Servers darstellt.

[37] Oracle, 05.05.2015, Kapitel 48.2.4, http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#GKBAA
[38] Oracle, 05.05.2015, Kapitel 48.3, http://docs.oracle.com/javaee/7/tutorial/security-webtier003.htm
[39] Oracle, 05.05.2015, http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html

Erläuterung Anhand eines Beispiels: (HttpServlet)

```
public class TestServlet extends HttpServlet {

// Bearbeite HTTP GET und POST Methoden

    protected void processRequest(HttpServletRequest request,

            HttpServletResponse response)

            throws ServletException, IOException {

            response.setContentType("text/html;charset=UTF-8");

            PrintWriter out = response.getWriter();
```

5.2.2 HttpServletRequest[40]

Das HttpServletRequest Objekt dient zur Repräsentation der Browser Anfrage. Mit dem HttpServletRequest Objekt können alle Informationen die der Browser schickt gelesen werden. Es werden verschiedene Methoden bereitgestellt die zur Authentisierung dienen.

Login und Logout

Alternativ zu der Deklarativen Form basierenden Authentisierung, können auch die *login* und *logout* Methoden verwendet werden. Die Login erwartet Dabei die Parameter Benutzername und Passwort als String. Diese Parameter werden mit der Methode *getParameter* beschafft(nur für einzelne Werte. Für mehrere Werte ist die Methode *getParameterValues* zuständig).

Erläuterung Anhand eines Beispiels: (Login und Logout)

```
//Code von der Klasse Testservlet

try {

            String userName = request.getParameter("txtUserName");

            String password = request.getParameter("txtPassword");

            try {

               request.login(userName, password);

            } catch(Servlet Expection ex) {

// weitere Code
```

[40] Oracle, 05.05.2015, http://docs.oracle.com/javaee/6/api/javax/servlet/ServletRequest.html

Authenticate

Die *authenticate* Methode erlaubt der Anwendung einen Authentisierungs Prozess zu starten, dieser Prozess ist im ServletContext definiert[41]. Er kann zum Beispiel dem Benutzer eine Eingabebox anibeten, wo er seinen Benutzernamen und sein Passwort eingeben kann.

Erläuterung Anhand eines Beispiels: (Authenticate)

```
//Code von der Klasse Testservlet

try {

            request.authenticate(response);

} catch // weiterer Code
```

ServletContext[42]

Das ServletContext Objekt enthält Meta Informationen über die Web Anwendung, zum Beispiel die Parameter in der web.xml Datei.

Methoden zur Identifizierung des Users[43]

Zur Identifizierung des User gibt es verschiede Möglichkeiten.

getRemoteUser

Die Methode *getRemoteUser*, liefert den Namen zurück mit dem der User angemeldet ist. Ist kein User angemeldet liefert die Methode null zurück.

isUserInRole

Die Methode *isUserInRole* erwartet als Parameterwert einen String, mit dem die Rolle des Users verglichen werden soll. Dazu *security-role-ref* Element kann im Deployment Descriptor genutzt werden um Referenzen für die Rollen darzustellen(Es ist aber nicht unbedingt nötig diese zu nutzen).

security-role-ref

Das security-role-ref Element ist eine Referenz zum Vergleichen der Rollen Namen, die von der Anwendung per *isUserInRole* Methode genutzt werden und der Rollen Namen die im Deployment Descriptor festgelegt wurden. Dies bringt gewisse Freiheiten in der Namensgebung der Rollen mit sich.

[41] Oracle, 05.05.2015, Kapitel authenticate
http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html#authenticate(javax.servlet.http.HttpServletResponse)
[42] Oracle, 05.05.2015, http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html
[43] Oracle, 05.05.2015, Beschreibungen der Methoden
http://docs.oracle.com/javaee/6/api/javax/servlet/ServletRequest.html

Erläuterung Anhand eines Beispiels: (*security-role-ref* Element)

```
<servlet>

...

    <security-role-ref>

        <role-name>cust</role-name>

        <role-link>bankCustomer</role-link>

    </security-role-ref>

...

</servlet>



<security-role>

    <role-name>bankCustomer</role-name>

</security-role>
```

Mit dem *security-role-ref* Elemnt kann nun die Methode *isUserInRole* mit dem Parameterwert „cust“ aufgerufen werden.

getUserPrincipal

Die Methode *getUserPrincipal* gibt den Namen des Principals des Benutzers aus.

5.2.3 HttpServletResponse[44]

Das HttpServletResponse Objekt ist für die Beantwortung der Browseranfragen zuständig. Das HttpServlerResponse Objekt hat unter anderen folgenden wichtigen Methoden.

setContentType

Um den Browser mitzuteilen welche Art von Dokument zu erwarten ist, muss das durch die Methode *setContentType* geregelt werden.

getWriter

Falls es sich bei dem Dokument um ein HTML Dokument handelt muss noch ein Schreiber erzeugt werden. Hierfür. wird die Methode *getWriter* aufgerufen.

Im Beispiel (HttpServlet(Kapitel 5.2.1)) sind diese beiden Methoden in den letzten beiden Zeilen zu finden.

[44] Oracle, 06.05.2015, http://docs.oracle.com/javaee/6/api/javax/servlet/ServletResponse.html

6. Sicherheit auf Enterprise JavaBeans Ebene(EJB)

Auf der logischen Ebene(EJB Ebene) gibt es ebenfalls deklarative sowie programmatische Sicherheitsmaßnahmen die ergriffen werden können, um bestimmte Ressourcen von unbefugten Zugriff zu schützen.

6.1 Deklarative Sicherheit

6.1.1 Wichtige Annotations[45]

Auf der EJB Ebene wird die deklarative Sicherheit durch Annotations festgelegt. Die wichtigsten sind hierbei:

@DeclareRoles

Die DeclareRoles Annotation muss über einer Klasse gesetzt werden. Diese bestimmt welche Rollen in dieser Klasse definiert sind. Diese Annotation ist allerding nur optional, da es nicht notwendig ist die Rollen zu definieren bevor sie genutzt werden[46]

```
@DeclareRoles({„admin“ „user“})

public class test { // Rest vom Code
```

@RolesAllowed

Die RolesAllowed Annotation darf über Klassen und Methoden gestellt werden. Hierbei wird festgestellt, dass nur die definierten Rollen Zugriff auf die Klasse oder Methode haben.

```
//Code von Beispiel @DeclareRoles

@RolesAllowed({„admin“})

public void adminMethod() { }// Rest vom Codes
```

@PermitAll

Die PermitAll Annotation wird wie die RolesAllowed Annotation verwendet. Nur dürfen hier alle Rollen auf die Klasse oder Methode zugreifen.

@DenyAll

Die DenyAll Annotation ist der Gegenspieler zur PermitAll Annotation und verbietet den Zugriff auf die Klasse oder Methode durch eine beliebige Rolle.

Bemerkung: Annotations über Methoden bestimmen deren Zugriffsrechte, auch wenn in der Klasse allgemein andere Zugriffsrechte gelten.

[45] Oracle, 06.05.2015, Tabelle 22-2 Security Annotation, https://docs.oracle.com/html/E13981_01/servsecr004.htm

[46] JavaBeat, 06.05.2015, Überschrift: Declares Roles annotation, http://www.javabeat.net/securing-ejb-applications/

6.1.2 Die ejb-jar.xml Datei[47]

Die ejb-jar.xml Datei ist ein Deployment Descriptor der folgende Eigenschaften festlegen kann:

- Festlegung der Pflichtfelder
- Zuweisung der Rollen
- Einen optionalen Namen für die ejb-client-jar Datei
- Einen optionalen Assembly Descriptor

Rollen in der ejb-jar.xml Datei

Rollen werden in der ejb-jar.xml Date unter dem Element *security-role* mit dem Element *role-name* definiert. Optional kann mit dem *descripton* Element die Rolle näher beschrieben werden.

Erläuterung Anhand eines Beispiels: [48](*security-role* Element)

```
<!– - Ein einfacher Ausschnitt aus einer ejb-jar.xml Datei.-->

<ejb-jar>

        <!-- … -->

        <assembly-descriptor>

                <security-role>

                        <description> Eine einfache Role </description>

                        <role-name> Die Rolle </role-name>

                </security-role>

        </assembly-descriptor>

</ejb-jar>
```

Die anderen Elemente, der ejb-jar.xml Datei, können dies Rollen durch das Benutzen des Elements *role-name* zugreifen.

[47] Oracle, 06.05.2015, Überschrift: What ist he ejb-jar.xml File
http://docs.oracle.com/cd/E16439_01/doc.1013/e13981/undejdev003.htm#CHDHAACJ
[48] Redhat, 06.05.2015, Überschrift Example 1.3 ejb-jar.xml descriptor fragment,
https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/5/html/Security_Guide/J2EE_Declarative_Security_Overview-Security_roles.html

6.2 Programmatisch Sicherheit[49]

Auf der EJB Ebene, kann auch durch Programmatische Sicherheit für die Authentisierung und Autorisierung gesorgt werden.

Dafür werden einige Komponenten benötigt. Die Instanz SessionContext speichert die zur Laufzeit ausgeführte Session ab[50].

Mit diesen SessionContext kann nun mit der Methode getCallerPrincipal der aktuelle Benutzer festgestellt werden. Die Methode liefert ein WLSUserPrincipal zurück.

Das Objekt SessionContext, biete auch die Möglichkeit, die Rolle des aktuellen Benutzers zu überprüfen. Hierfür wird die Methode isCallerInRole verwendet, diese Erwartet als Parameter ein String.

Erläuterung Anhand eines Beispiels:[51]

```
//benötigte Imports

@Declare Roles(„User“)

public class testBean

@Resource SessionContext ctx;



@RolesAllowed(„User“)

public testMethod() {

Principal callerPrincipal = ctx.getCallerPrincipal();

if (ctx.getCallerInRole(„User“)) { // Rest vom Code
```

[49] Oracle, 07.05.2015, Kapitel: Using Programmatic Security With EJBs, http://docs.oracle.com/cd/E11035_01/wls100/security/ejb_client.html#wp1028164
[50] Oracle, 07.05.2015, http://docs.oracle.com/javaee/7/api/javax/ejb/SessionContext.html
[51] Oracle, 07.05.2015, Kapitel: Modyfing ConverterBean, http://docs.oracle.com/cd/E19798-01/821-1841/bncaa/index.html

7. Fazit

Es wurde deutlich, dass die Java Enterprise Edition, einige Sicherheitsmechanismen aufweist. Es gibt die Möglichkeit auf der Web Anwendungsebene, sowie auf der Enterprise JavaBean Ebene zu integrieren.

Um die Ressourcen eines Unternehmens erfolgreich zu schützen, ist es somit notwendig einige Sicherheitsmechanismen zu implementieren. Dabei wird zwischen den Ebenen und der Art der Sicherheit unterschieden.

Auf der Ebene der Web Anwendung, sollten die folgende Mittel genutzt werden, da diese nicht auf der EJB Ebene umzusetzen sind.

- Sicherung einer vertrauensvollen Verbindung zwischen Client und Server
- Ressourcen mit ähnlichen Sicherheitsregeln zusammenfassen
- Geeigneten Authentisierungsmechanismus auswählen

Es empfiehlt sich bei Anwendungen mit vielen vertraulichen Ressourcen die deklarative Sicherheit zu wählen, solange diese ausreicht, weil hier die geschützten Ressourcen kompakt in einer Datei enthalten sind. Ansonsten muss die programmatische Sicherheit hinzugezogen werden.

Die Festlegung der Rollen kann, in beiden Ebenen stattfinden und sollte je nach Gebrauch, auch dort definiert werden. Auch hier sollte bei vielen vertraulichen Ressourcen die Deklarative Sicherheit bevorzugt werden.

Deshalb sollte meiner Meinung nach, die Deklarative Sicherheit zum Großteil benutzt werden, und die Programmatische, nur wenn es wirklich notwendig ist.

Das Thema Beidseitige Authentisierung wird als Zeittechnischen Gründen nicht behandelt.

8. Literaturverzeichnis

Oracle Java EE Tutorial

http://docs.oracle.com/

Stackoverflow

http://stackoverflow.com/

CodeJava

http://www.codejava.net/

Bsi Bund

https://www.bsi.bund.de/