Hochschule

für angewandte Wissenschaften

München

Fakultät für Informatik und Mathematik

Seminararbeit

Titel, deutsch:

"Sicherheit in unternehmenskritischen Applikationen"

Titel, englisch:

"Security in business critical applications"

Prüfer:	Theis Michael
Verfasser:	Leidner Michael
Studiengruppe:	IB 6A
Abgabedatum:	10.06.2016

Erklärung:
Hiermit erkläre ich, dass ich die Arbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

10.06.2016 

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IIOP	Internet Inter-ORB Protocol
IT	Informationstechnik
JAAS	Java Authentication and Authorization
Java EE	Java Enterprise Edition
Java GSS	Java Generic Security Services
Java SE	Java Standard Edition
JCE	Java Cryptography Extension
JMX	Java Management Extensions
JSF	JavaServer Faces
JSSE	Java Secure Sockets Extension
MAC	Message Authentication Code
URL	Uniform Resource Locator
SASL	Simple Authentication and Security Layer
SSL	Secure Sockets Layer
TLS	Transport Layer Security
XML	Extensible Markup Language

1 Einführung

Diese Seminararbeit behandelt das Thema „Sicherheit in unternehmenskritischen Applikationen“. Dabei stellen sich in diesem Zusammenhang zwei Fragen. Die Eine ist die Frage danach, was Sicherheit ist. Die andere Frage ist die nach dem, was eine unternehmenskritische Applikation ist.

„Laut Definition muss eine unternehmenskritische Anwendung verfügbar sein, damit die Mitarbeiter ihrer Arbeit nachgehen können, Partner mit dem Unternehmen zusammenarbeiten können und Kunden Produkte und Services erwerben können. Wenn eine unternehmenskritische Anwendung auch nur für einige Minuten offline ist, verliert das Unternehmen Geld, Markenwert und das Vertrauen seiner Kunden und Partner." (EMC 2010, S. 6)

Damit die Verfügbarkeit von diesen Anwendungen gewährleistet ist, müssen geeignete Sicherheitsmaßnahmen ergriffen werden. Nach Witt ist IT-Sicherheit die „Einhaltung bestimmter Sicherheitsstandards, die die Verfügbarkeit, Unversehrtheit oder Vertraulichkeit von Informationen betreffen, durch Sicherheitsvorkehrungen in oder bei der Anwendung von informationstechnischen Systemen/Komponenten". (Witt 2012, S. 20) Jedes Unternehmen hat also sensible Ressourcen, die von mehreren Usern und Ressourcen verwendet werden und die es zu schützen gilt. Oracle stellt für die Sicherheit in Java EE einige Mechanismen und Frameworks zur Verfügung.

In dieser Arbeit werden zuerst die Grundlagen und der Aufbau der Sicherheit im Sinne von Java EE erläutert. Anschließend wird darauf eingegangen an welchen Stellen eine Java EE Anwendung, mit welchen Mitteln, gehärtet werden kann. Dabei wird auf das Sichern von Webanwendungen und das Sichern von Enterprise Beans eingegangen. Im Anschluss wird beschrieben, wie sich eigene Sicherheitsmechanismen in eine Java EE Umgebung integrieren lassen. Zu Letzt gibt es ein abschließendes Fazit.

2 Grundlagen der Sicherheit im Sinne von Java EE

2.1 Funktion von Sicherheitsmechanismen

Bevor auf die eigentlichen Sicherheitsmechanismen eingegangen wird, ist es wichtig zu wissen, welche Funktionen die Implementierung von Sicherheitsmechanismen bieten sollten und JavaEE bietet:

- Schutz vor unbefugtem Zugriff auf Anwendungsfunktionen und geschäftlichen oder persönlichen Daten
- Mechanismen für die Nachvollziehbarkeit für die von Usern ausgeführten Operationen
- Schutz vor Ausfällen und Unterbrechungen, die sich auf die Qualität auswirken
- Zudem sollten implementierte Sicherheitsmechanismen einfach zu verwalten sein, Transparent für den User und interoperabel über Anwendungs- und Unternehmensgrenzen hinweg

(vgl. Oracle 2014a)

2.2 Eigenschaften der Java EE Anwendungssicherheit

Eigenschaften der Anwendungssicherheit dienen dazu, wenn sie richtig eingesetzt werden, die Sicherheitsbedrohungen zu minimieren.

- Authentifizierung: Das Mittel, mit denen Kommunikationseinheiten, wie Client und Server die Identität, für die Autorisierung, feststellen.
- Autorisierung / Zugriffskontrolle: Die Autorisierung dient dazu festzustellen, ob ein User die Berechtigung hat auf Ressourcen zuzugreifen oder Operationen auszuführen. Dies dient dem Zweck der Datenintegrität, dem Datenschutz oder auch der Verfügbarkeit von Daten.
- Datenintegrität: Damit überprüft werden kann, dass die Daten nicht von Dritten verändert wurden.
- Vertraulichkeit / Datenschutz: Damit nur autorisierte Benutzer vertrauliche Daten einsehen können.
- Unbestreitbarkeit: Ein Mittel damit nachzuvollziehen ist, das Benutzer Aktionen durchgeführt haben.
- Servicequalität: Ein Mittel für die Verbesserung des Services über verschiedene Technologien.

- Auditierung: Aufzeichnung von sicherheitsrelevanten Ereignissen, damit die Wirksamkeit von Sicherheitsmaßnahmen ermittelt werden kann.

(vgl. Oracle 2014a)

2.3 Securing Containers

Java EE Anwendungen bestehen aus einzelnen Komponenten. Diese Komponenten liegen in verschiedenen Containern. Die Sicherheit für die Komponenten wird in Java EE von den Containern bereitgestellt. Ein Container stellt dabei zwei Arten der Sicherheit zur Verfügung, zum Einem die deklarative Sicherheit und zum Anderen die programmatische Sicherheit. (vgl. Oracle 2014a; Oracle 2014b)

Bei der programmatischen Sicherheit werden Sicherheitsmaßnahmen direkt in der Anwendung im Programmcode eingebettet und wird dann genutzt, wenn die deklarativen Sicherheitsmaßnahmen nicht ausreichen. Im Gegensatz zur programmatischen Sicherheit wird bei der deklarativen Sicherheit nicht innerhalb des Programmcodes gearbeitet, sondern es werden Annotations und Deployment Descriptors genutzt. (vgl. Oracle 2014a; Oracle 2014b)

Ein Deployment Descriptor ist eine XML-Datei, die außerhalb der Anwendung liegt und eine grundlegende Sicherheitsstruktur der Anwendung, wie z. B. Rollen, Zugriffskontrollen und Authentifizierung, ausdrückt. Deployment Desciptors bieten den Vorteil, dass Änderungen vorgenommen werden können, ohne den Quellcode zu ändern. Annotations können innerhalb einer Java-Klasse angegeben werden und enthalten Informationen zur Sicherheit. Wenn die Anwendung bereitgestellt wird, können die Informationen von dem Deployment Descriptor genutzt werden oder von dieser überschrieben werden. Mit Hilfe von Annotations können die Informationen zur Sicherheit direkt beim Code angeben werden und müssen nicht Descriptors verwenden. Allerdings können nicht alle Sicherheitsinformationen als Annotations angegeben werden. Diese müssen in Deployment Descriptors angegeben werden. (vgl. Oracle 2014a; Oracle 2014b)

2.4 JavaEE Sicherheitsmechanismen

Neben den zwei Arten der Java EE Sicherheit sind die Sicherheitsmechanismen von Java EE nach drei Ebenen aufgeteilt: Application-Layer Security, Transport-Layer Security und Message-Layer Security.
Bei der Application-Layer Security sind die Komponenten Container für die Bereitstellung von Sicherheitsmechanismen verantwortlich. Die Application-Layer Security dient dem Schutz der Anwendung und ist wie eine Firewall für die Anwendung. Die Firewall schützt den Kommunikationsstrom und die Anwendungsressourcen. Der Vorteil der Application-Layer Security ist, dass sie genau auf die Anwendung und deren Bedürfnisse zugeschnitten ist und anwendungsspezifische Einstellungen möglich sind. Allerdings hat diese Art der Sicherheit auch mehrere Nachteile. Die Anwendung besitzt Sicherheitsattribute, die nicht zwischen verschiedenen Anwendungstypen übertragbar sind. Wenn mehrere Protokolle unterstützt werden, gefährdet das die Sicherheit. Die Daten sind also nur so lange geschützt, wie sie in der Anwendung sind. Wenn die Daten z. B. für einen Webservice verwendet werden, dann ist für den Schutz der Daten die Application-Layer Security nicht ausreichend und es werden die Transport-Layer Security und die Message-Layer Security benötigt. (vgl. Oracle 2014d)

Transport-Layer Security wird für die sichere Kommunikation zwischen Client und Provider verwendet. Für die Kommunikation wird auf HTTPS (HyperText Transfer Protocol Secure) in Verbindung mit SSL (Secure Sockets Layer) gesetzt. Eine Transport-Layer Security kann für die Authentifizierung und Nachrichtenintegrität verwendet werden. Dabei können Client und Server erst eindeutige Schlüssel als kryptografisches Mittel austauschen, bevor sie Daten hin und her senden. Mit dieser Methode sind die Daten vom Client bis zur Ankunft bei der Anwendung geschützt. Dies hat den Vorteil, dass es eine standardisierte Technologie ist, die einfach zu handhaben ist und einen Schutz für die gesamt Nachricht hat. Der letzte Punkt stellt dabei aber auch den Nachteil dieser Sicherheitsebene dar. Der Sicherheitsmechanismus kennt den Inhalt der Nachricht nicht, weshalb nicht einzelne Inhalte der Nachricht ausgewählt und besonders geschützt werden können. Außerdem bietet dieser Sicherheitsmechanismus keinen Schutz über den Transport der Nachricht hinweg. Dies bedeutet, dass die Nachricht bei der Ankunft beim Empfänger den Schutz verliert. Es stellt also keine Ende zu Ende Lösung dar. Damit einzelne Elemente der Nachricht geschützt werden, braucht man die Message-Layer Security. (vgl. Oracle 2014d)

Die Message-Layer Security wird genutzt, um Nachrichten vom Sender zum Empfänger verschlüsselt zu übermitteln. Im Gegensatz zum Mechanismus der Transport-Layer Security kann die Nachricht nur vom Empfänger entschlüsselt werden, auch wenn die Nachricht über mehrere Knoten geschickt wird. Des Weiteren können auch einzelne Bestandteile der Nachricht verschlüsselt werden, damit diese von einem Zwischenknoten entschlüsselt werden können, weil z. B. diese Teilnachricht für einen Zwischenknoten gedacht ist, während der Rest der Nachricht für einen anderen Empfänger bestimmt ist und von dem Zwischenknoten nicht entschlüsselt werden kann. Message-Layer Security hat den Vorteil, dass die Sicherheit der Nachricht über alle Zwischenknoten bestehen bleibt und für Teile der Nachricht selektiv angewendet werden kann. Zusätzlich ist die Sicherheit unabhängig von der Anwendungsumgebung und dem Transportprotokoll. Allerdings hat der Mechanismus die Nachteile, dass er komplex ist und der Prozess aufwendiger ist. (vgl. Oracle 2014d)

2.5 Secure Sockets Layer

Wie bereits im vorherigen Kapitel beschrieben, wird für die Transport-Layer Security die SSL-Technologie verwendet, für die sichere und verschlüsselte Versendung von Daten zwischen Server und Client.

SSL sorgt dafür, dass beim ersten Kommunikationsversuch der Server dem Client mitteilt, wer er ist. Dies geschieht in Form eines Zertifikates. Der Server kann auch vom Client ein Zertifikat verlangen, um zu zeigen, wer er ist. Des Weiteren sorgt SSL dafür, dass Daten die innerhalb eines Netzwerkes versendet werden, nicht unterwegs von Dritten, die diese Daten lesen können, geändert werden und nicht von Dritten entschlüsselt werden. Des Weiteren werden SSL-Antworten verschlüsselt. Die Verschlüsselung und Entschlüsselung verbraucht viel Rechenleistung. Deshalb ist es von Vorteil, wenn nur Informationen die wirklich verschlüsselt werden müssen, verschlüsselt werden. Dies sind z. B. persönliche Informationen, wie Name, Adresse oder Zahlungsinformationen. Damit SSL verwendet werden kann, muss einfach bei der Adresse „https“ anstatt „http“ verwendet werden. (vgl. Oracle 2014f)

2.6 JavaSE Sicherheitsmechanismen

JavaEE nutzt neben den eigenen Sicherheitsmaßnahmen auch die Sicherheitsmechanismen von JavaSE.

Zu den JavaSE Sicherheitsmechanismen zählen Java Authentication and Authorization Service (JAAS), Java Generic Security Services (Java GSS), Java Cryptography Extension (JCE), Java Secure Sockets Extension (JSSE) und Simple Authentication and Security Layer (SASL). (vgl. Oracle 2014d)

Java Authentication and Authorization besteht aus mehreren APIs, die Dienste für die Authentifizierung und die Zugriffskontrolle bieten. JAAS bietet ein erweiterbares Framework für die programmatische Benutzerauthentifizierung und Autorisierung. Java Generic Security Services ist eine tokenbasierte API, die genutzt werden kann, um Nachrichten zwischen Anwendungen auszutauschen. (vgl. Oracle 2014d)
Java GSS hat den Vorteil, dass mit der Nutzung der API Programmierer einen einheitlichen Zugriff auf Sicherheitsdienste bekommen. Java Cryptography Extension ist ein Framework und enthält Implementierungen für die Verschlüsselung, Schlüsselgenerierung, Schlüsselvereinbarung und Message Authentication Code (MAC) Algorithmen. Es stehen verschiedene Verfahren für die Verschlüsselung zur Verfügung, wie symmetrisch, asymmetrisch, Block- und Stromchiffren. (vgl. Oracle 2014d)
Java Secure Sockets Extension ist ein Framework und enthält eine Java-Version von Secure Sockets Layer (SSL) und Transport Layer Security (TLS) Protokollen und hat Funktionen für die Datenverschlüsselung, Serverauthentifizierung, Nachrichtenintegrität und Clientauthentifizierung. JSSE ist somit für eine sichere Kommunikation im Internet geeignet. (vgl. Oracle 2014d)
Simple Authentication and Security Layer ist ein Protokoll für die Authentifizierung und bietet die Möglichkeit für die Einrichtung einer Sicherheitsschicht zwischen Client und Server. SASL gibt an, wie Authentifizierungsdaten ausgetauscht werden. (vgl. Oracle 2014d)

2.7 Securing GlassFish Server

Neben der eigentlichen Security in JavaEE ist auch die Sicherheit des Servers dafür verantwortlich, wie sicher eine Anwendung ist. Für die Verwendung von JavaEE wird deshalb von Oracle als Server der GlassFish von Oracle empfohlen. GlassFish ist ein Open-Source-Anwendungsserver, welcher von Oracle gesponsert wird. Dies bedeutet, dass der GlassFish Server durch die Verbindung mit Oracle ideal auf JavaEE ausgelegt und abgestimmt ist.

Der GlassFish Server unterstützt das JavaEE Sicherheitsmodell. Er ist vollständig kompatibel und unterstützt verteilte Komponenten. GlassFish bietet verschiedene Konfigurationsmöglichkeiten für die folgenden Zwecke. Zum Beispiel das Hinzufügen, Löschen oder Ändern von autorisierten Benutzern und von bestehenden oder benutzerdefinierten Realms. Weitere Möglichkeiten des GlassFish Servers sind das Konfigurieren von sicheren HTTP und Internet Inter-ORB Protocol (IIOP) Listeners und von sicheren Java Management Extensions (JMX). (vgl. Oracle 2014c)
Ein weiterer Zweck ist das Definieren von Interfaces für die Autorisierung mittels JACC. GlassFish bietet auch die Möglichkeit für Anpassungen an den Authentifizierungsmechanismen. Darüber hinaus können Einstellungen an den Richtlinienberechtigungen gesetzt und geändert werden. (vgl. Oracle 2014c)

2.8 Users, Groups, Roles und Realms

Neben den in den vorangegangenen Punkten dargelegten Sicherheitsmechanismen arbeitet JavaEE mit Users, Groups, Roles und Realms.

User sind in JavaEE entweder einzelne Individuen oder repräsentieren ein Anwendungsprogramm. Ein User ist eindeutig identifizierbarer. Diese User können in Groups zusammengefasst werden. User in einer Group haben gleiche Merkmale, die sie klassifizieren. Die Verwendung von Groups macht es leichter den Zugang von einer großen Menge an Usern zu verwalten. Roles geben an, auf welche Ressourcen man in einer Anwendung zugreifen darf. User und oder Groups können den Roles zugeordnet werden, aber auch mehrere Roles besitzen. Ein wesentlicher Unterschied zwischen Groups und Roles ist der, dass die Zuteilung zu Groups für den gesamten Server oder für Realms gelten. Roles gelten dagegen nur für Anwendungen. Mit Realms können Ressourcen auf dem

Server in Bereiche zusammengefasst werden, für die dann zugewiesene User und Groups gültig sind und die ein eigenes Authentifizierungsschema und eine eigene Autorisierungsdatenbank besitzen. (vgl. Oracle 2014e)

Abbildung 1 zeigt den Zusammenhang zwischen Roles, Groups und Usern.

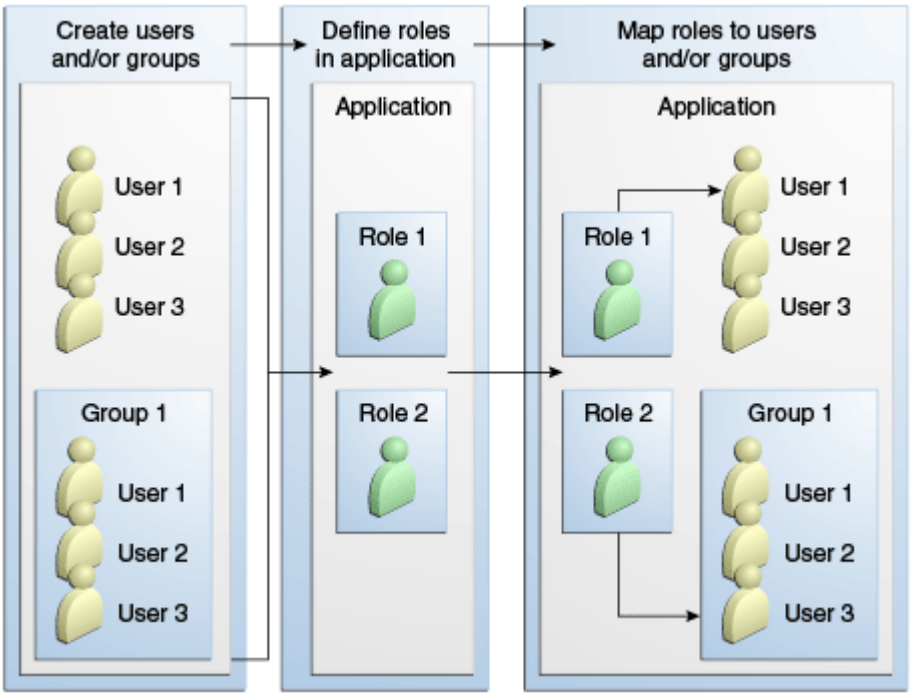

Abbildung 1: User, Group, Role (Quelle: Oracle 2014e)

3 Webanwendungen sichern

JavaEE Anwendungen sind nach einem Model mit mehreren Ebenen aufgebaut. Die Webanwendungen laufen in der Webebene. Webanwendungen können durch deklarative oder programmatische Sicherheitsmaßnahmen geschützt werden. Java Webkomponenten können JavaServer Faces (JSF) oder Java Servlets sein. (vgl. Oracle 2014g; Oracle 2014h)

3.1 Deklarative Sicherheit

Für die deklarative Sicherheit bei Webanwendungen stehen in JavaEE, wie bereits im vorherigen Kapitel erwähnt, Annotations und Deployment Descriptors zur Verfügung. Die durch Annotations deklarierten Werte können durch Konfigurationen im Deployment Descriptor überschrieben werden. Angaben im Deployment Descriptor sind höher zu bewerten als Sicherheitsangaben mittels Annotations. (vgl. Oracle 2014h)

3.1.1 Sicherheitseinschränkungen

Beim Thema Sicherheitseinschränkungen geht es darum, den Zugriff auf Ressourcen zu beschränken. Sicherheitseinschränkungen können, wenn Servlets genutzt werden, mittels Annotations angegeben werden. Für diesen Zweck stehen `@HttpConstraint`, `@HttpMethodConstraint` und `@ServletSecurity` zur Verfügung. (vgl. Oracle 2014i)

Im Beispiel wird durch die Annotation dafür gesorgt, dass nur User mit der Rolle „Admin“ dieses Servlet nutzen können.

Beispiel:

```
1   @ServletSecurity(@HttpConstraint(rolesAllowed={"admin"}))
2   public class AdminServlet extends HttpServlet {
3       ...
4   }
```

Wenn keine Servlets genutzt werden, müssen Sicherheitseinschränkungen im Deployment Descriptor angegeben werden. Dafür steht das `<security-constraint>` Element mit den Unterelementen `<web-resource-collection>`, `<auth-constraint>` und `<user-data-constraint>` zur Verfügung. (vgl. Oracle 2014i)

Das Element `<web-resource-collection>` schränkt den Zugriff auf angegebene

URL Bereiche und die Verwendung von HTTP-Methoden ein. Innerhalb des Unterelementes `<url-pattern>` können Teile einer angeforderten URL angegeben werden, die geschützt werden sollen. In den beiden Unterelementen `<http-method>` und `<http-method-omission>` können HTTP-Methoden angegeben werden die geschützt oder nicht geschützt werden sollen. (vgl. Oracle 2014i)

`<auth-constraint>` enthält die Rollen, die Zugriff auf die geschützten Ressourcen bekommen. Das Unterelement `<role-name>` besitzt die Bezeichnung einer Rolle. (vgl. Oracle 2014i)

Mit dem `<user-data-constraint>` Element kann angegeben werden, wie die Daten zwischen Client und Server transportiert werden sollen, z. B. über HTTP oder HTTPS. Im Unterelement `<transport-guarantee>` können die Werte `CONFIDENTIAL`, `INTEGRAL` oder `NONE` angegeben werden. (vgl. Oracle 2014i)

Im Beispiel bekommen ausschließlich User mit der Rolle „admin“ und der Rolle „manager“ Zugriff auf Ressourcen, die im Bereich „/adminBereich/“ liegen. Außerdem sind nur die HTTP-Methoden „GET“ und „DELETE“ geschützt. Der Wert „INTEGRAL“ im Beispiel sorgt dafür, dass die Daten beim Transport nicht geändert werden können.

Beispiel:

```
1    <security-constraint>
2       <web-resource-collection>
3            <url-pattern>/adminBereich/*</url-pattern>
4            <http-method>GET</http-method>
5            <http-method>DELETE</http-method>
6       </web-resource-collection>
7       <auth-constraint>
8            <role-name>admin</role-name>
9            <role-name>manager</role-name>
10      </auth-constraint>
11      <user-data-constraint>
12           <transport-guarantee>INTEGRAL</transport-guarantee>
13      </user-data-constraint>
14  </security-constraint>
```

3.1.2 Authentifizierungsmechanismus

Der Authentifizierungsmechanismus identifiziert einen User. Wird ein Authentifizierungsmechanismus spezifiziert, wird der User authentifiziert, bevor er Zugang zu Ressourcen bekommt, die durch Sicherheitseinschränkungen beschränkt werden. Der Mechanismus wird für alle geschützten Ressourcen einer Anwendung verwendet. Die folgenden Authentifizierungen stehen zur Verfügung: Basic, Form-based, Digest, Client und Mutual. Basic- und Form-based-Authentifizierung sind keine sicheren Authentifizierungsmechanismen. Allerdings kann durch die Verwendung von sicheren Transportverbindungen die Sicherheit der beiden erhöht werden. (vgl. Oracle 2014i)

Der Basic-Mechanismus verlangt vom Browser, nach den Namen und dem Passwort vom User zu fragen, wenn dieser auf geschützte Ressourcen zugreifen will. Die gesendeten Daten vom User werden mit einer Datenbank mit autorisierten Usern verglichen und der Server schickt bei Erfolg die angeforderten Ressourcen an den Client zurück. Der Basic-Mechanismus ist der voreingestellte Mechanismus. (vgl. Oracle 2014i)
Innerhalb des `<login-config>` Elementes wird der Authentifizierungsmechanismus mit den entsprechenden Unterelementen angegeben. Im `<auth-method>` wird die Authentifizierungsmethode angegeben.

Beispiel:

```
1   <login-config>
2       <auth-method>BASIC</auth-method>
3   </login-config>
```

Der Form-based-Mechanismus bietet die Möglichkeit eine Log-in-Seite, sowie eine Fehlerseite anzuzeigen. Sobald der Benutzer auf geschützte Ressourcen zugreifen will, wird er vom Server zur Log-in-Seite weitergeleitet. Der Client übermittelt dann die ausgefüllte Form und nach der Überprüfung der Daten gelangt dieser entweder zu der von ihm angeforderten Seite oder zu der Fehlerseite. (vgl. Oracle 2014i)
`<form-login-config>`, `<form-login-page>` und `<form-error-page>` enthalten die Informationen für die Log-in-Seite und die Fehlerseite. Damit die Form-based Authentifizierung richtig funktioniert, sollte die Action der Form `j_security_check` sein.

Beispiel Deployment Descriptor:

```
1   <login-config>
2       <auth-method>FORM</auth-method>
3       <form-login-config>
4           <form-login-page>/LoginSeite.xhtml</form-login-page>
5           <form-error-page>/Fehlerseite.xhtml</form-error-page>
6       </form-login-config>
7   </login-config>
```

Beispiel HTML-Seite:

```
1   <form method="POST" action="j_security_check">
2       <input type="text" name="username">
3       <input type="password" name="password">
4   </form>
```

Die Digest-Authentifizierung funktioniert von der Benutzung her wie die Basic-Authentifizierung. Allerdings versendet der Client das Passwort als Hash zur Verschlüsselung. (vgl. Oracle 2014i)

Beispiel:

```
1   <login-config>
2       <auth-method>DIGEST</auth-method>
3   </login-config>
```

Die Clientauthentifizierung nutzt das Public-Key-Zertifikat vom Client, damit der Server diesen authentifizieren kann. Der Mechanismus nutzt HTTPS dafür, dieser ist dadurch sicherer als die Basic- oder Form-based-Authentifizierung. (vgl. Oracle 2014j)

Beispiel:

```
1   <login-config>
2       <auth-method>CLIENT-CERT</auth-method>
3   </login-config>
```

Beim Mutual-Mechanismus authentifizieren sich Client und Server gegenseitig. Dafür gibt es zwei Arten, zum Einen mittels Zertifikat und zum Anderen mittels Name und Passwort. Bei der Variante mit Zertifikaten stellt der Client eine Anfrage. Der Server stellt dem Client sein Zertifikat zur Überprüfung bereit. Wenn die Überprüfung erfolgreich ist, stellt der Client sein Zertifikat dem Server bereit. Ist dies ebenfalls erfolgreich, dann bekommt der Client Zugriff auf die Ressourcen. Die Variante mit Name und Passwort verläuft genauso wie die mit dem Zertifikat. Der einzige Unterschied ist, dass der Client für seine Überprüfung dem Server Name und Passwort übermittelt. Es gibt zwei Möglichkeiten, um den Mutual-Mechanismus zu verwenden. Zum Einem wie im Beispiel unten über den Deployment Descriptor und zum Anderen über das `<clientAuth>` Element im `<certificate>` Realm, das auf `true` gesetzt werden muss. (vgl. Oracle 2014j)

Beispiel:

```
1   <login-config>
2       <auth-method>CLIENT-CERT</auth-method>
3   </login-config>
```

3.1.3 Sicherheitsrollen deklarieren

Um die Sicherheit zu erhöhen und um Sicherheitseinschränkungen richtig nutzen zu können, werden Rollen benötigt. Diese können im Deployment Descriptor angegeben werden. Dazu werden die Elemente `<security-role>` und `<role-name>` benötigt. Im Deployment Descriptor können alle Rollen aufgelistet werden, die in der Anwendung genutzt werden. (vgl. Oracle 2014i)

Beispiel:

```
1   <security-role>
2       <role-name>admin</role-name>
3   </security-role>
4   <security-role>
5       <role-name>manager</role-name>
6   </security-role>
```

3.2 Programmatische Sicherheit

Will man die Authentifizierung programmatisch implementieren, bietet JavaEE dafür das Interface „HttpServletRequest“ an. Das Interface enthält die `authenticate`, `login` und `logout` Methoden. Die authenticate-Methode erlaubt der Anwendung, eine Authentifizierung für den Benutzer durchzuführen. Beim Benutzer erscheint eine Dialogbox, in der der Username und das Passwort angegeben werden können. Mit der login-Methode können Username und Passwort gespeichert werden. Während die logout-Methode die Informationen zurücksetzt. (vgl. Oracle 2014k)

Darüber hinaus bietet das Servlet 3.1 die Möglichkeit, auf weitere Sicherheitsinformationen des Aufrufers zuzugreifen. Die Methoden `getRemoteUser`, `isUserInRole` und `getUserPrincipal` stehen dafür bereit. `GetRemoteUser` liefert den Benutzernamen des Aufrufers. Falls es keinen authentifizierten Benutzer gibt, wird `null` zurückgegeben. `IsUserInRole` überprüft, ob der Benutzer, die als Parameterwert angegebene Rolle, besitzt oder nicht, und gibt einen boolean Wert zurück. `GetUserPrincipal` bestimmt den principal-Namen und gibt ein `java.security.Principal` Objekt zurück. Falls dieser nicht gefunden wird, wird `null` zurückgegeben. (vgl. Oracle 2014k) Ein Principal kann eine Person, ein Unternehmen, eine Anwendung, etc. sein. Die Methode `getUserPrincipal` gibt also nicht unbedingt den eigentlichen Aufrufer am Client zurück. (vgl. Oracle 2014l)

Beispiel:

```
1   public class LoginServlet extends HttpServlet {
2      protected void processRequest(HttpServletRequest request,
3      HttpServletResponse response)
4      throws ServletException, IOException {
5         try {
6            out.println(request.isUserInRole("admin")+"<br>");
7            out.println(request.getRemoteUser()+"<br>");
8            out.println(request.getUserPrincipal()+"<br>");
9
10           try {
11              request.login(userName, password);
12
13              out.println(request.isUserInRole("admin")+"<br>");
14              out.println(request.getRemoteUser()+"<br>");
15              out.println(request.getUserPrincipal()+"<br>");
16           } catch(ServletException ex) {
17              out.println("Login Fehlgeschlagen");
18              return;
19           }
20        } finally {
21            request.logout();
22             out.close();
23        }
24     }
25  }
```

4 Enterprise Beans sichern

Enterprise Beans Komponenten sind in der Java multiebenen Architektur in der Geschäftsebene. (vgl. Oracle 2014m) Wie bei den Webkomponenten können Enterprise Beans Komponenten durch deklarative oder programmatische Sicherheitsmaßnahmen geschützt werden.

4.1 Deklarative Sicherheit

Annotations für Enterprise Beans Komponenten können für eine Klasse, eine Methode einer Klasse oder für beides angegeben werden. Für den Fall, dass beide Varianten verwendet werden, wird die Berechtigung die für die gesamte Klasse gilt, durch die Berechtigung, die bei der Methode angegeben wird, überschrieben. (vgl. Oracle 2014n)

Mit `@DeclareRoles` können die Rollen für die Klasse deklariert werden, welche im Anschluss durch die Methode `isCallerInRole` (programmatisch) getestet werden können. Mit der `@RolesAllowed`-Annotation wird angegeben, welche Rollen die Klasse oder die Methode verwenden dürfen. Die in `@DeclareRoles` und in `@RolesAllowed` definierten Rollen sind alle Rollen, die die Anwendung verwendet. In beiden Annotationen können mehrere Rollen angegeben werden. Wenn alle definierten Sicherheitsrollen Zugriff auf eine Methode oder Klasse haben sollen, wird die Annotation `@PermitAll` benutzt. Die gegenteilige Annotation ist die `@DenyAll`. Diese Annotation dient dazu, dass keine definierten Sicherheitsrollen die Methode verwenden dürfen. Sie kann ausschließlich für Methoden verwendet werden. (vgl. Oracle 2014n)

Im Beispiel werden die Rollen „Admin“, „Manager“ und „User“ deklariert. Zugriff auf die Klasse „testKlasse“ bekommen Benutzer mit der Rolle „Admin“. Auf die Methode „test1“ bekommt ebenfalls nur der „Admin“ Zugriff. Während auf „test2“ der „Manager“ und der „User“ Zugriff haben und nicht der „Admin“. Bei „test3“ haben alle Benutzer Zugriff und bei „test4“ keiner.

Beispiel:

```
1   @DeclareRoles(„Admin", „Manager", „User")
2   @RolesAllowed(„Admin")
3   public class testKlasse {
4       public void test1 (){
5            ...
6       }
7       @RolesAllowed(„Manager", „User")
8       public void test2 (){
9            ...
10      }
11      @PermitAll
12      public void test3 (){
13           ...
14      }
15      @DenyAll
16      private void test4 (){
17           ...
18      }
19  }
```

4.2 Programmatische Sicherheit

Für die programmatische Sicherheit bietet JavaEE das `javax.ejb.EJBContext` Interface an, um auf Informationen der Identität des Aufrufers zuzugreifen. Dafür bietet das Interface zwei Methoden an, zum Einen die Methode `isCallerInRole` und zum Anderen die Methode `getCallerPrincipal`. Die programmatische Sicherheit kann z. B. eingesetzt werden, um innerhalb einer Methode eine Beschränkung einzuführen. Indem nur ein Teil des Codes ausgeführt wird, wenn der Aufrufer eine bestimmte Rolle innehat. (vgl. Oracle 2014n)

Die Methode `isCallerInRole` erwartet eine zu überprüfende Sicherheitsrolle als Parameter und gibt entweder `true` bei erfolgreicher Überprüfung zurück oder ein `false`. `GetCallerPrincipal` liefert das principal-Objekt, das den Aufrufer identifiziert, zurück. (vgl. Oracle 2014n)

Im Beispiel wird geprüft, ob der Aufrufer die Rolle „Admin“ hat. Wenn ja wird die Methode changeAddress() aufgerufen und das principal-Objekt geholt um den Namen des Aufrufers zu bekommen. Mit diesem als Schlüssel soll dann der User gefunden und die Adresse geändert werden.

Beispiel:

```
1   public class testKlasse {
2       SessionContext sc;
3       EntityManager em;
4
5       public void testRole(...) {
6            if (!sc.isCallerInRole("admin")) {
7                 this.changeAddress(...)
8            }else{
9                 ...
10           }
11      }
12
13      private void changeAddress(...) {
14           callerPrincipal = ctx.getCallerPrincipal();
15           callerKey = callerPrincipal.getName();
16
17           User user = em.find(User.class, callerKey);
18
19           user.setAddress(...);
20      }
21  }
```

5 Weitere Sicherheitsmaßnahmen

Die Sicherheitsmechanismen, die von JavaEE bereitgestellt werden, sind wichtig, um ein Grundgerüst an Sicherheit in Anwendungen vorzufinden. Allerdings muss der Entwickler auch eigene Sicherheitsmaßnahmen ergreifen oder zumindest nach bestimmten Kriterien arbeiten.

Ein Beispiel hierfür ist die Datenvalidierung. Hier kann es eine Möglichkeit sein, den Ansatz zu verfolgen, Whitelisting dem Blacklisting vorzuziehen. Dies bedeutet, dass, anstatt bestimmte Muster herauszufiltern, bestimmte Muster nur erlaubt sind. (vgl. BSI 2006, S. 24) Ein weiterer Ansatz ist, das Gegenüberstellen von Input-Validierung und Output Validierung. Während die Input-Validierung Schutz vor z. B. Injection bilden soll, soll die Output-Validierung das Cross-Site Scripting verhindern. (vgl. BSI 2006, S. 20) Generell sollten die Input-Daten nicht vom Client geprüft werden, sondern vom Server. Validierungen per JavaScript können z. B. vom User verändert werden. (vgl. BSI 2006, S. 32)

Das SQL-Injection sollte ebenfalls verhindert werden. Dies kann durch das Verwenden von Prepared Statements anstelle von dynamischen SQL-Statements erreicht werden. (vgl. BSI 2006, S. 29)

Ein weiterer Punkt ist die Kontrolle von URL-Weiterleitungen. Es ist ratsam nur bekannte Links weiterzuleiten, Weiterleitungen über Zwischenseiten zu machen, Weiterleitungen einzuschränken und Header-Manipulationen zu verhindern. (vgl. BSI 2006, S. 38 f.) POST-Requests sollten den GET-Requests vorgezogen werden. Dies kann Cross-Site Scripting verhindern. (vgl. BSI 2006, S. 56)

Es gibt noch viele weitere Maßnahmen, die ergriffen werden können, z. B. sichere Passwörter verlangen, Minimalitätsprinzip für Informationen, Session Management, Verhindern von automatisierten Angriffen und vermeiden einer hohen Fehlertoleranz.

6 Fazit

JavaEE bietet viele Sicherheitsmechanismen und eine gute Architektur, um Anwendungen zu schützen und wie schon im vorherigen Punkt gesagt, ein gutes Gerüst. Die JavaEE Sicherheitsmechanismen lassen sich oft auch schnell und einfach durch die Nutzung von Annotations und Deployment Descriptor Files umsetzen. Darüber hinaus bietet JavaEE gute Möglichkeiten, programmatische Erweiterungen einzubauen. Die Mechanismen sind auf die Komponenten Architektur und die Ebenen Architektur ausgelegt und bieten sowohl in der Webebene wie auch in der Geschäftsebene einen guten Schutz.

Allerdings bietet der Aufbau der Mechanismen nur einen Schutz, wenn diese auch korrekt, von den dafür zuständigen Entwicklern umgesetzt werden und durch weitere Maßnahmen erweitert werden. Die JavaEE Sicherheit bietet vor allem einen Schutz und eine gute Technologie für die Authentifizierung, Autorisierung, den Transport vom Server zum Client und der Kommunikation zwischen den Komponenten. Sicherheitsmaßnahmen, die darüber hinaus gehen, müssen von den Entwicklern selber implementiert oder darauf geachtet werden, darunter z. B. die Validierung, das vermeiden von Injections oder Cross-Site Scripting.

Am Ende jedoch Helfen die besten Sicherheitsmaßnahmen nichts, wenn die Verantwortlichen für die Anwendungen diese nicht auch umsetzen und sich bereits im Vorhinein damit auseinandersetzen und sich Gedanken darüber machen wie es am besten und mit welchen Mitteln umgesetzt werden kann. Jede Anwendung hat für sich eigene entscheidende Kriterien die eine Rolle dabei spielen können.

Deshalb möchte ich diese Arbeit mit einem Zitat von dem amerikanischen ehemaligen Hacker Kevin Mitnick beenden:

> „Die Organisationen stecken Millionen von Dollars in Firewalls und Sicherheitssysteme und verschwenden ihr Geld, da keine dieser Maßnahmen das schwächste Glied der Sicherheitskette berücksichtigt: Die Anwender und Systemadministratoren.“

Quellen-/Literaturverzeichnis

BSI 2006: „Sicherheit von Webanwendungen - Maßnahmenkatalog und Best Practices“ [Online] BSI Abgerufen unter: <https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/WebSec/WebSec_pdf.pdf?__blob=publicationFile> [Abrufdatum: 25.05.2016]

EMC 2010: „Die Virtualisierung geschäftskritischer Anwendungen - Technologiekonzepte und geschäftliche Überlegungen“ [Online] EMC Abgerufen unter: <http://germany.emc.com/collateral/software/white-papers/h6859-virtzng-business-crtcl-appt s-wp.pdf> [Abrufdatum: 10.04.2016]

ORACLE 2014a: „Java Platform, Enterprise Edition: The Java EE Tutorial - 47.1 Overview of Java EE Security“ [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/security-intro001.htm#BNBWK> [Abrufdatum: 11.04.2016]

ORACLE 2014b: „Java Platform, Enterprise Edition: The Java EE Tutorial - 47.3 Securing Containers“ [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/security-intro002.htm#BNBWY> [Abrufdatum 11.04.2016]

ORACLE 2014c: „Java Platform, Enterprise Edition: The Java EE Tutorial - 47.4 Securing GlassFish Server" [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/security-intro004.htm#BNBXI> [Abrufdatum 12.04.2016]

ORACLE 2014d: „Java Platform, Enterprise Edition: The Java EE Tutorial - 47.2 Security Mechanisms" [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/security-intro002.htm#BNBWY> [Abrufdatum 13.04.2016]

ORACLE 2014e: „Java Platform, Enterprise Edition: The Java EE Tutorial - 47.5 Working with Realms, Users, Groups, and Roles" [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/security-intro005.htm#BNBXJ> [Abrufdatum 13.04.2016]

ORACLE 2014f: „Java Platform, Enterprise Edition: The Java EE Tutorial - 47.6 Establishing a Secure Connection Using SSL" [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/security-intro006.htm#BNBXW> [Abrufdatum 17.04.2016]

ORACLE 2014g: „Java Platform, Enterprise Edition: The Java EE Tutorial - 48 Getting Started Securing Web Applications" [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/security-webtier.htm#BNCAS> [Abrufdatum 22.05.2016]

ORACLE 2014h: „Java Platform, Enterprise Edition: The Java EE Tutorial - 48.1 Overview of Web Application Security" [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/security-webtier001.htm#BNCAT> [Abrufdatum 22.05.2016]

ORACLE 2014i: „Java Platform, Enterprise Edition: The Java EE Tutorial - 48.2 Securing Web Applications" [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/security-webtier002.htm#GKBAA> [Abrufdatum 22.05.2016]

ORACLE 2014j: „Java Platform, Enterprise Edition: The Java EE Tutorial - 50.2 Authentication Mechanisms" [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/security-advanced002.htm#GLIEN> [Abrufdatum 22.05.2016]

ORACLE 2014k: „Java Platform, Enterprise Edition: The Java EE Tutorial - 48.3 Using Programmatic Security with Web Applications" [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/security-webtier003.htm#GJIIE> [Abrufdatum 22.05.2016]

ORACLE 2014l: „Interface Principal" [Online] Abgerufen unter: <https://docs.oracle.com/javase/7/docs/api/java/security/Principal.html> [Abrufdatum 23.05.2016]

ORACLE 2014m: „Java Platform, Enterprise Edition: The Java EE Tutorial - 1.3 Distributed Multitiered Applications" [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/overview003.htm#BNAAY> [Abrufdatum 24.05.2016]

ORACLE 2014n: „Java Platform, Enterprise Edition: The Java EE Tutorial - 49.2 Securing Enterprise Beans" [Online] Abgerufen unter: <http://docs.oracle.com/javaee/7/tutorial/security-javaee002.htm#BNBYL> [Abrufdatum 25.05.2016]

WITT Bernhard 2012: „Grundlagen der Datenschutzes und der IT-Sicherheit (Teil 1c)“ [Online] Abgerufen unter: <https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui/datenschutz/VL2012-1c.pdf> [Abrufdatum: 11.04.2016]