



integrata
cegos

Java und Spring Security

Javacream

Einführung

- Java-Anwendungen sind wie alle anderen Software-Programme anfällig gegen typische Hacker-Angriffe:
 - Verarbeitung unvalidierter und damit potenziell gefährdender Benutzereingaben,
 - Ausführen von Code aus nicht vertrauenswürdigen Quellen,
 - Ausspionieren von Informationen durch Abhören unsicherer Kommunikationskanäle,
 - Unzulässige Ausführung von Programmteilen durch eine fehlerhafte Implementierung einer Authentifizierungs- und Autorisierungs-Routine.
- Die dadurch hervorgerufenen Schäden können selbst bei trivial scheinenden Lücken immens sein.
 - Es ist deshalb obligatorisch, zumindest die typischen Security-Lücken von vornherein auszuschließen und alle anderen durch ständige Qualitätssicherung zu erkennen und zu beheben.

- Die OWASP-Gruppe stellt jedes Jahr unter anderem eine Liste der „Top-10“ der Security-Probleme dar.



- Die Platzierung innerhalb der Liste wird durch eine Kombination aus Häufigkeit und Gefährlichkeit getroffen.

- SQL Injection ist ein Begriff, der wohl den meisten Entwicklern, die zumindest rudimentär auf das Thema Security sensibilisiert sind, bekannt sein dürfte.
- Die Security-Lücke besteht darin, dass unvalidierte Benutzereingaben zu einem SQL-Skript aufbereitet werden, so dass der Angreifer potenziell die vollständige Kontrolle über den gesamten Datenbestand erhält.
- `ResultSet resultSet = statement.executeQuery("select message from messages where user = '"+ userCriterion + "'");`
- Der rot dargestellte Teil zeigt die Security-Lücke durch das Hinzufügen des Kriterien-Ausdrucks zum SQL-Statement.

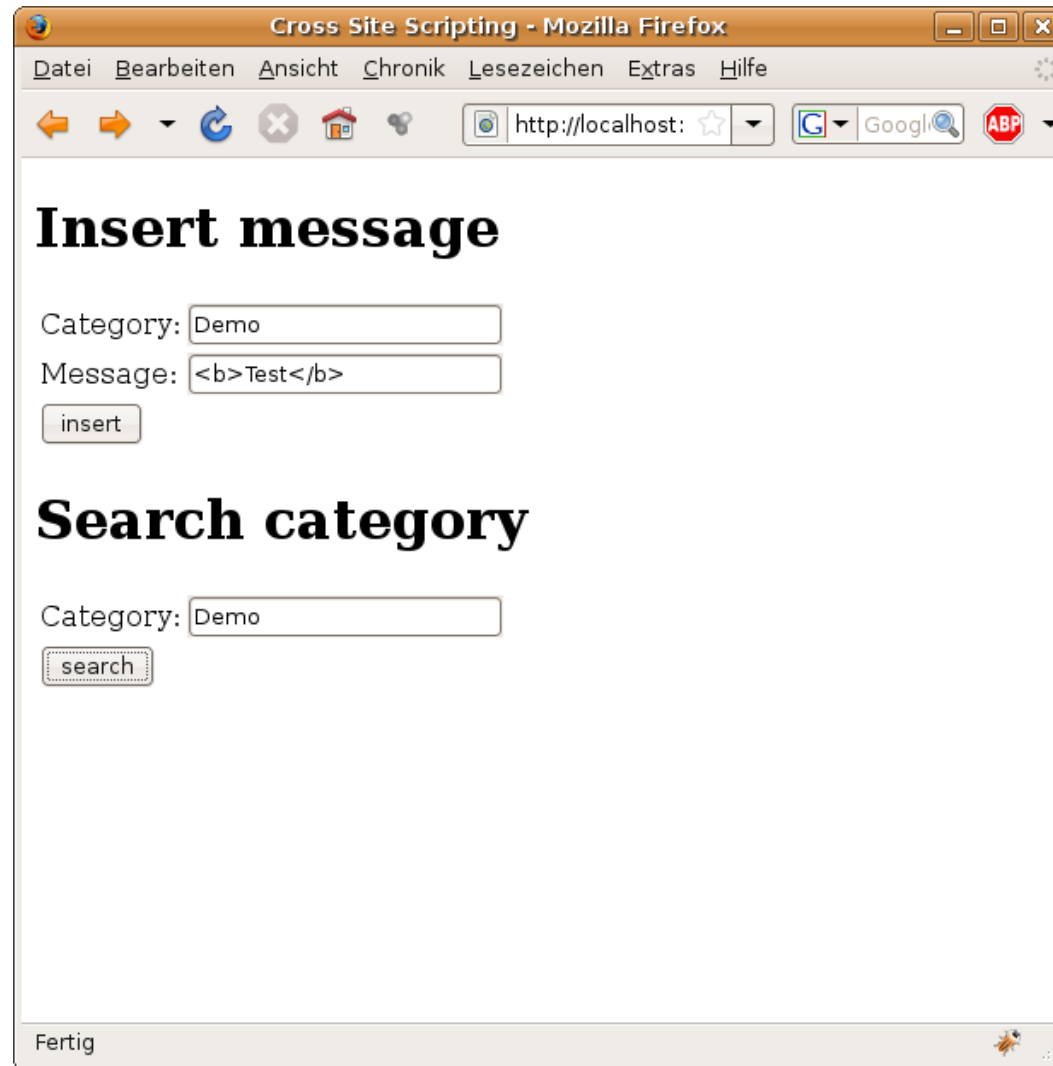
- Einige Attacken:
 - `String userCriterion = "User-0' or '1' = '1";`
 - `String userCriterion = "User-0' INSERT INTO MESSAGES VALUES ('A fake message', 'User-0') SELECT * from MESSAGES where user='User-0";`
 - `String userCriterion = "User-0' SELECT ADMINNAME from ADMINNS where 'h'='h";`

- Häufig wird unterschätzt, dass der Einsatz eines O/R-Mappers wie beispielsweise Hibernate oder dem Java Persistence API vor der SQL-Injection schützt.
- Dies ist aber nur beschränkt der Fall
- `Query query = entityManager.createQuery("from MessageHolder as message where message.user='" + userCriterion + "'");`
- Auch hier wird wie im Beispiel vorher im rot dargestellten Teil eine Benutzereingabe direkt zu einer Abfrage, diesmal in der JPA Query-Sprache, umgesetzt.
- Potenzielle Attacken:
 - `String userCriterion = "User-0' or 'h'='h";`
 - `String userCriterion = "User-0' AND (select count(*) from Admin) > 0 AND 'h' = 'h";`
 - `String userCriterion = "User-0' AND (select count(*) from Admin as admin where admin.adminName LIKE 'A%') > 0 AND 'h' = 'h";`

- Wird die Abfragelogik in Stored Procedures abgelegt, so wird gerne vergessen, dass damit eine Abfrage- oder Darstellungs-Logik nur in die Datenbank verschoben wird.
- Damit muss selbstverständlich auch die Parameter-Prüfung innerhalb der Stored Procedure erfolgen.
- Ist dies nicht der Fall sind auch Stored Procedures für SQL Injection-Attacken anfällig.

- Die aktuell laut der OWASP-Organisation empfindlichste Security-Lücke ist das Cross Site Scripting. Das Grund-Szenarium dafür lautet:
 - Eine Anwendung erlaubt es einem beliebigen (oder auch durchaus authentifiziertem!) Benutzer, Informationen zu hinterlegen.
 - Diese Informationen können von anderen Benutzern betrachtet bzw. allgemein verwendet werden.
 - Bei der Benutzung dieser Informationen werden Vorgänge ausgelöst, die für den Benutzer schädlich sein können.
- Das Paradebeispiel hierfür sind Web 2.0-Anwendungen, bei denen der Austausch beliebiger Informationen über Benutzer hinweg ja gerade das kennzeichnende Feature darstellt:
 - Chat-Rooms, Online-Foren oder Gästebücher sind typische Anwendungen.
- Es ist an dieser Stelle jedoch definitiv zu beachten, dass eine Beschränkung auf Web-Anwendungen das wahre Bedrohungspotenzial verschleiert.
 - Auch eine Administrationsoberfläche für eine Datenbank, die Datensätze darstellt, ist dafür anfällig.
 - Selbst ein simpler Editor für Log-Dateien kann theoretisch für Attacken benutzt werden.

Cross Site Scripting: Eine anfällige Seite



Cross Site Scripting - Mozilla Firefox

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

http://localhost: Google ABP

Insert message

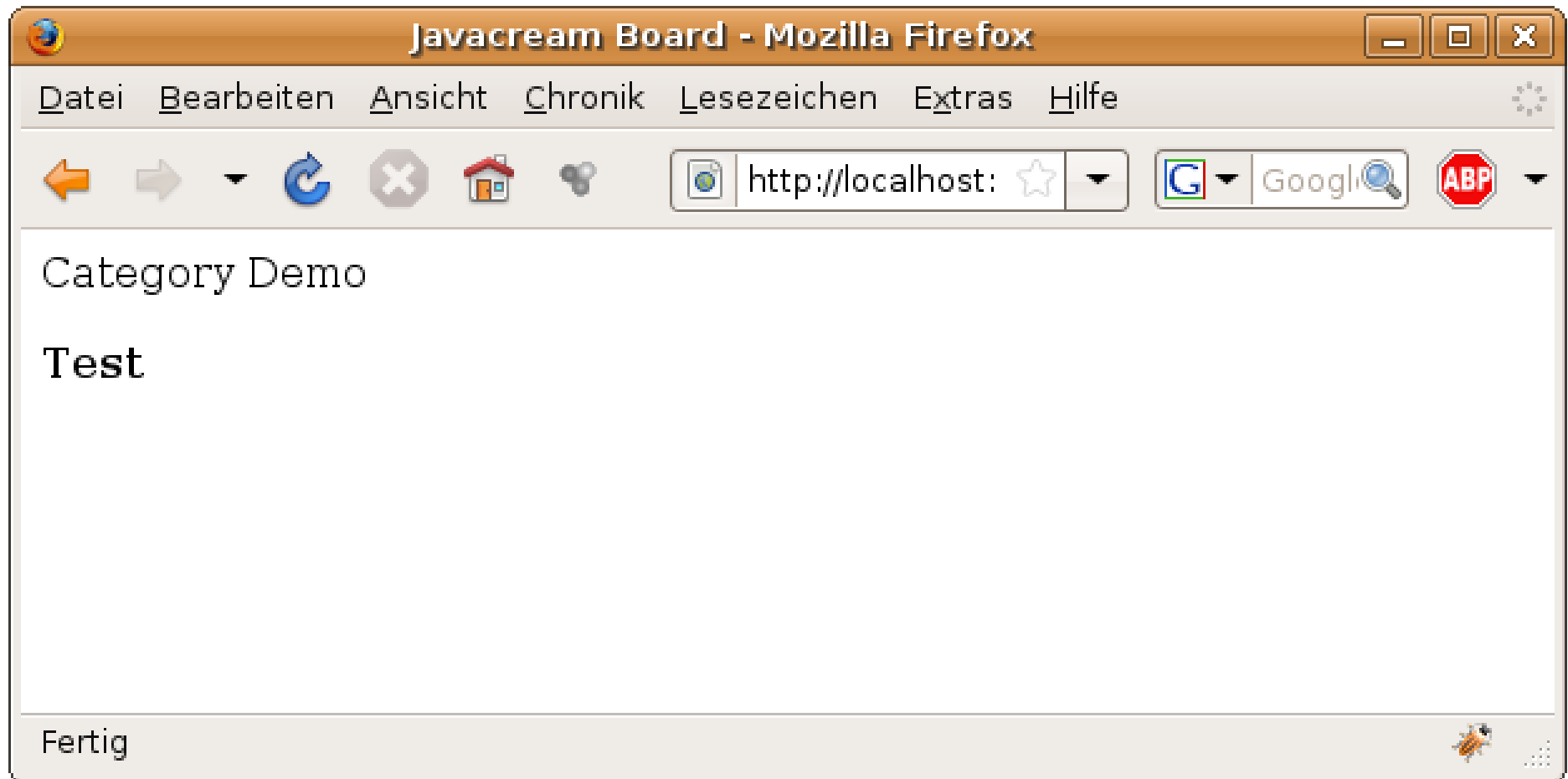
Category:

Message:

Search category

Category:

Fertig



- Links auf unsichere Seiten
- Integration von Javascript
- Verlinken von Ressourcen (z.B. JPG-Bilder), die Fehler in Plug-Ins (z.B. das JPG-PlugIn im Internet Explorer) ausnutzen
- ...

- Diese Sicherheitslücke nutzt in seiner klassischen Form aus, dass in der URL eines Aufrufs eine Dateiangabe einer Ressource angegeben ist.
 - Der Angreifer kann diese Angabe manipulieren und auf diese Art und Weise den Zugriff auf kritische Informationen erhalten.
- Typische Beispiele sind:
 - Auslesen von Konfigurations- oder Log-Dateien.
 - Auslesen von uncompilierten jsp-Seiten.
 - ...
- In der modernen Welt der Java-Applikationen kann „Insecure Direct Object Reference“ anderes interpretiert werden, nämlich als „echte“ Referenz auf ein Business Objekt.
 - Frameworks wie Apache Struts oder auch Web Services Frameworks zeichnen sich unter anderem dadurch aus, dass rein konfiguratativ durch eine Deskriptor-Datei praktische jede Klasse ohne zusätzlichen Programmieraufwand für einen entfernten Methodenaufwurf freigegeben werden kann.

- Wo liegt das Problem?
- Ganz einfach:
 - Der von der Web Seite gesendete Request (hier ein GET-Request) ist für jeden einfach zu interpretieren.
 - So kann beispielsweise sofort statt des Methodennamens get ein remove ausprobiert werden.
 - Und schon kann der Client einen Eintrag löschen.
- Noch gefährlicher wird es dann, wenn auf Grund von Namenskonventionen der Angreifer andere Services erraten kann.
 - Wird diese Lücke nicht behoben ist damit der Zugriff auf die gesamte Domänenlogik möglich.
- Als Abhilfe dürfen nur die Methoden erreichbar sein, die wirklich nach Außen gegeben werden sollen.
 - Dies kann durch einen Servlet-Filter realisiert werden.

SECURITY UND JEE

Das Realm definiert einen eigenen Namensraum für Benutzer und Gruppen.

Ein Realm wird persistent verwaltet.

Ein einzelner Benutzer kann keiner, einer oder mehrerer Gruppen zugeordnet sein.

Benutzer und Gruppen können z.B. in Tabellen abgelegt werden.

Jede Gruppe besteht aus keinem, einem oder mehreren Benutzern.

Eine Rolle ist eine Menge aus Gruppen und

Benutzern.

■ Benutzer, Gruppen, Realms und Rollen
Rollen sind anwendungsabhängig.

<application>

...

<security-role>

<description>admin role</des

<role-name>admin</role-nam

</security-role>

<security-role>

<description>manager role</description>

<role-name>manager</role-name>

- Deklaration von Rollen auf der Application-Ebene

</security-role>

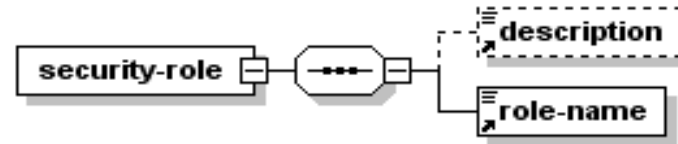
<security-role>

<description>everyone role</description>

<role-name>everyone</role-name>

</security-role>

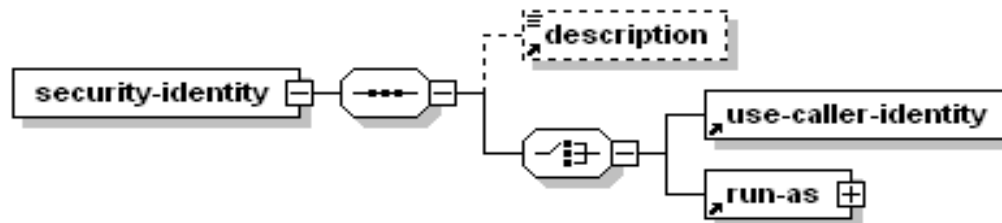
</application>



Generated with XMLSpy Schema Editor www.xmlspy.com

Zwei Alternativen pro EnterpriseBean in der ejb-jar.xml einstellbar:

run-as Angegebene Rolle verwenden.
use-caller-identity Aktuellen Benutzer verwenden.

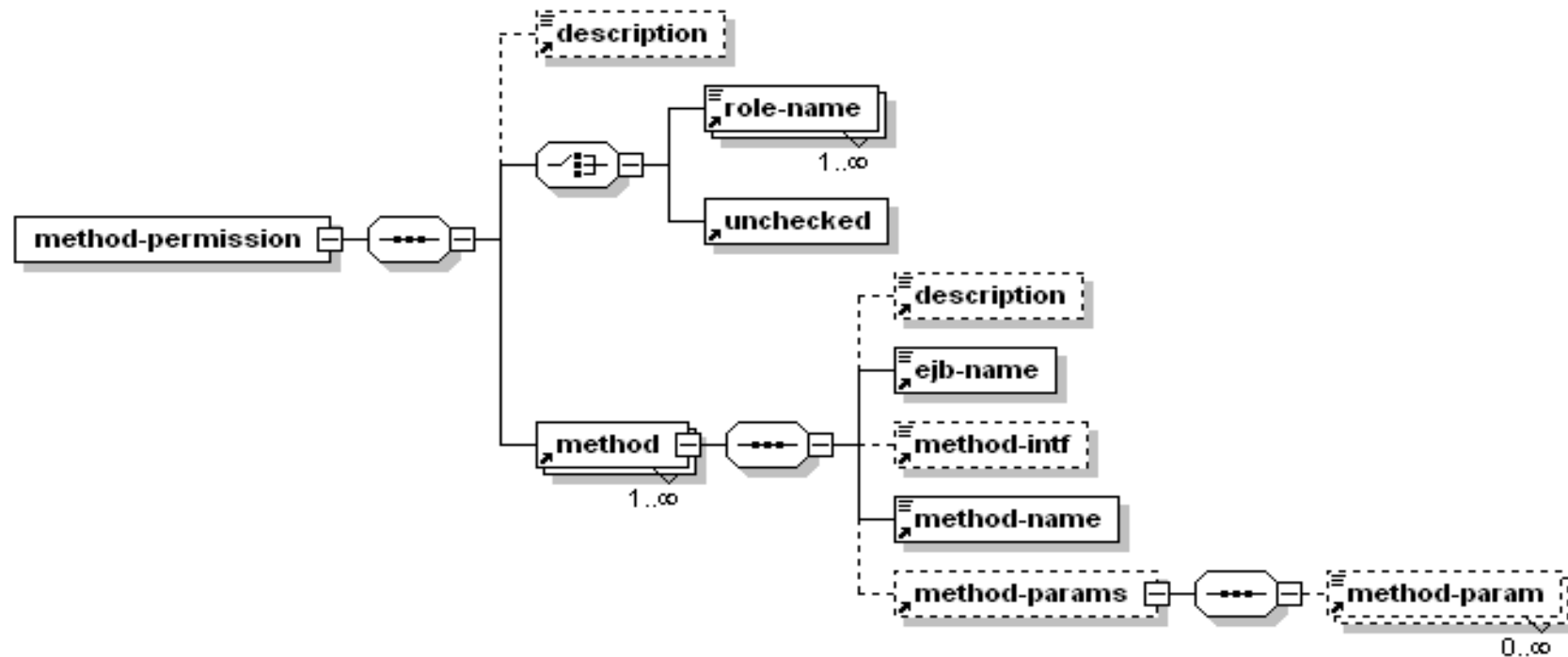


Generated with XMLSpy Schema Editor www.xmlspy.com

MessageDrivenBeans müssen „run-as“ verwenden.

- Das Messaging-Protokoll spezifiziert keinen Standard für die Übertragung des Subjects

Vergabe von Berechtigungen auf Methoden-Ebene



Generated with XMLSpy Schema Editor www.xmlspy.com

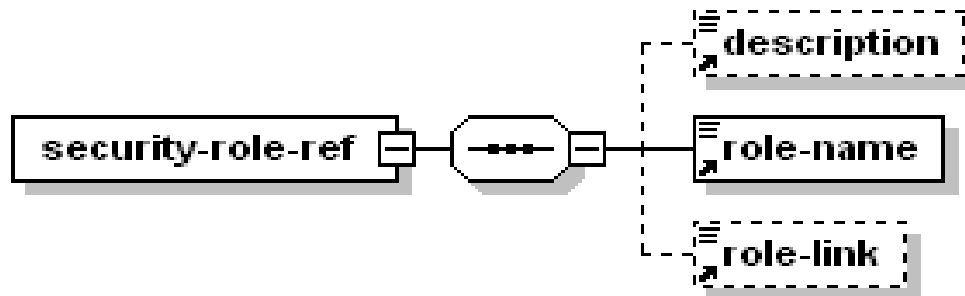
Obwohl die J2EE-Komponente keine Authentifizierungs-Logik enthalten soll, ermöglicht der Kontext den Zugriff auf bestimmte Anmelde-Informationen:

```
java.security.Principal getCallerPrincipal()
```

```
boolean isCallerInRole(String role)
```

Die Methode `isCallerInRole` verlangt noch ein Mapping des verwendeten Rollennamens mit vorhandenen Rollen
Role-Link-Mechanismus im Deskriptor.

- Methoden des `EJBContext`



Generated with XMLSpy Schema Editor www.xmlspy.com

JAAS

Prinzipal

Ein Name, eine Personalnummer, ein Benutzername.

Interface `java.security.Principal`

Credential

Ein Credential ist kein Java-Interface sondern irgendeine Klasse, die zur Authorisierung benutzt werden kann.

Je nach dem verwendeten Authentifizierungsmechanismus kann dies ein einfaches Byte-Array als Password oder ein Zertifikat sein.

- Grundbegriffe

Subject erfüllt drei Aufgaben:

Halten von einem oder mehreren Principal-Referenzen.
Halten von Credentials.

Ablegen eines konkreten Subjects innerhalb des aufrufenden Threads. Dies erfolgt durch den Aufruf einer der statischen Methoden `doAs()` oder `doAsPrivileged()`. Nach diesem Aufruf kann das Subject als Bestandteil des `AccessContext` ausgelesen werden.

- Das `javax.security.auth.Subject`

Definiert im Paket

`javax.security.auth.callback`

Benötigt ein Authentifizierungs-Mechanismus Informationen vom Client, wird im aufrufenden Thread eine Implementierung des CallbackHandler-Interfaces verwendet.

- Das Callba

interface
<i>javax.security.auth.callback.CallbackHandler</i>
<i>+ handle(<code>javax.security.auth.callback.Callback[]</code>):void</i>
!

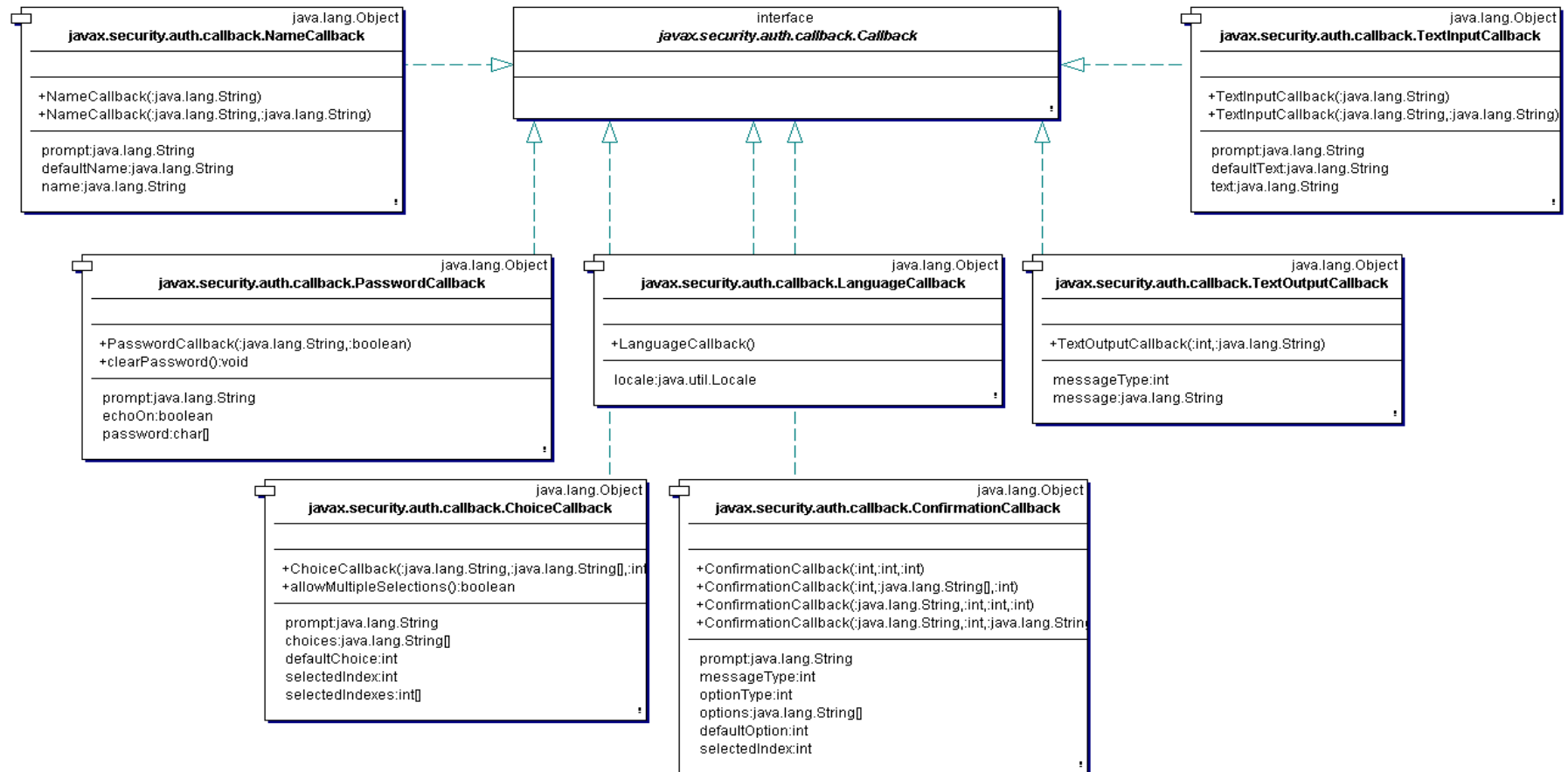
javax.security.callback.Callback ist ein
reines Marker-Interface

Callback-Implementierungen dienen zum
Sammeln und Transport von Informationen,
die zur Authentifizierung benötigt werden.

Der Authentifizierungs-Mechanismus ruft für den
CallbackHandler die Methode `handleCallback(Callback)`
mit geeigneten Callbacks auf.

Der CallbackHandler prüft den Typ des übergebenen
Callbacks und setzt die geforderten Parameter.

- Das Callback-Interface



class SimpleCallbackHandler implements
CallbackHandler {

public void handle(Callback[] callbacks)

throws IOException,

UnsupportedCallbackException{

for (int i = 0; i < callbacks.length; i++) {

if (callbacks[i] instanceof

TextOutputCallback) {

TextOutputCallback toc =

(TextOutputCallback)callbacks[i];

■ Ein einfacher CallbackHandler
switch (toc.getMessageType()) {

case

TextOutputCallback.INFORMATION:

System.out.println(toc.getMessage());

break;

//...

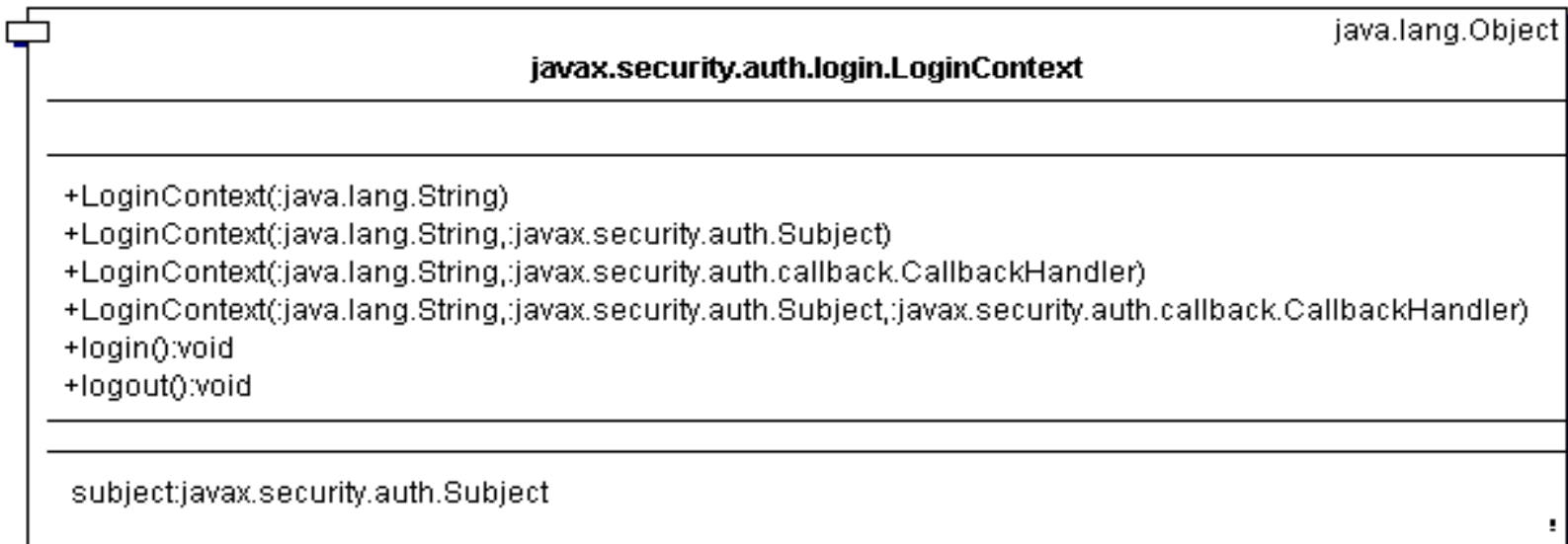
default: throw new

IOException("Unsupported message type: "

+ toc.getMessageType()).

Für den Aufrufenden vollkommen transparent durch Verwendung der Klasse `javax.security.login.LoginContext`

■ Erze



Der Authentifizierungs-Mechanismus wird durch das Interface `javax.security.auth.spi.LoginModule` abstrahiert.

■ C

interface <i>javax.security.auth.spi.LoginModule</i>	
<i>+initialize(javax.security.auth.Subject, javax.security.auth.callback.CallbackHandler, java.util.Map, java.util.Map):void</i> <i>+login():boolean</i> <i>+commit():boolean</i> <i>+abort():boolean</i> <i>+logout():boolean</i>	
	!

Der initialize-Methode werden alle benötigten Referenzen übergeben:

Das zu ergänzende Subject.

Der CallbackHandler des Aufrufenden.

Eine Map mit dem „Shared-State“, der allen an der Authentifizierung beteiligten LoginModules gemeinsam ist.

Eine Map mit den Konfigurationsparametern.

- Initialisierung des LoginModules

Die Implementierung des LoginModule-Interfaces ist Aufgabe des Anbieters eines JAAS-konformen Authentifizierungssystems.

Dazu kommen in der Regel noch spezielle Implementierungen des `java.security.Principal`-Interfaces sowie eine spezielle Credential-Klasse

Die Implementierung entscheidet, welche Informationen benötigt werden und erstellt eine Liste von Callback-Objekten, die dem LoginModule übergeben werden.

```
public class SimpleLoginModule
implements LoginModule {
//Attribute und initialize
public boolean login() throws
LoginException {
    Callback[] callbacks = new Callback[2];
    callbacks[0] = new
NameCallback(„Username: ");
    callbacks[1] = new
PasswordCallback(„Password: ", false);
    callbackHandler.handle(callbacks);
    username =
((NameCallback)callbacks[0]).getName();
    char[] tmpPassword =
((PasswordCallback)callbacks[1]).getPassw
ord();
    //...
```


Der Authentifizierungs-Teil des JAAS wird durch einen Provider zur Verfügung gestellt:

LoginModule,
Principal,
Credential-Klassen,
Spezielle Callback-Klassen,
Callback-Handler,

- Zusammenfassung:
Authentifizierungs-Mechanismus

Für den Client beschränkt sich der JAAS-Provider auf eine Hilfsklasse, die eine Konfigurations-Datei ausliest

`javax.security.auth.login.Configuration`

Eintrag in der Konfigurationsdatei der Java-Laufzeitumgebung für den Default-Provider Sun

`(<JAVA_HOME>\jre\lib\security\java.security)`

`login.configuration.provider=com.sun.security.auth.login.C`

`onfigFile`

■ Der JAAS-Client

Suchen einer Konfigurationsdatei an Hand eines Eintrags in der JRE-

Konfigurationsdatei:

`login.config.url.1=file:${user.home}/.java.login.config`

Auslesen einer Umgebungsvariable

`java.security.auth.login.config`

- Default-Konfiguration des JAAS

Syntax der Konfigurationsdatei: Authentifizierungs-Namen

```
SimpleLogin{  
  
};  
SSLLogin{  
  
};  
default{  
  
};
```

```
SimpleLogin{  
    com.hotspots.javax.security.login.spi.SimpleLoginModule;  
};  
SSLLogin{  
    com.hotspots.javax.security.login.spi.SSLLoginModule;  
    com.hotspots.javax.security.login.spi.CertificateLoginModule;  
};  
default{  
    com.hotspots.javax.security.login.spi.SimpleLoginModule;  
};
```

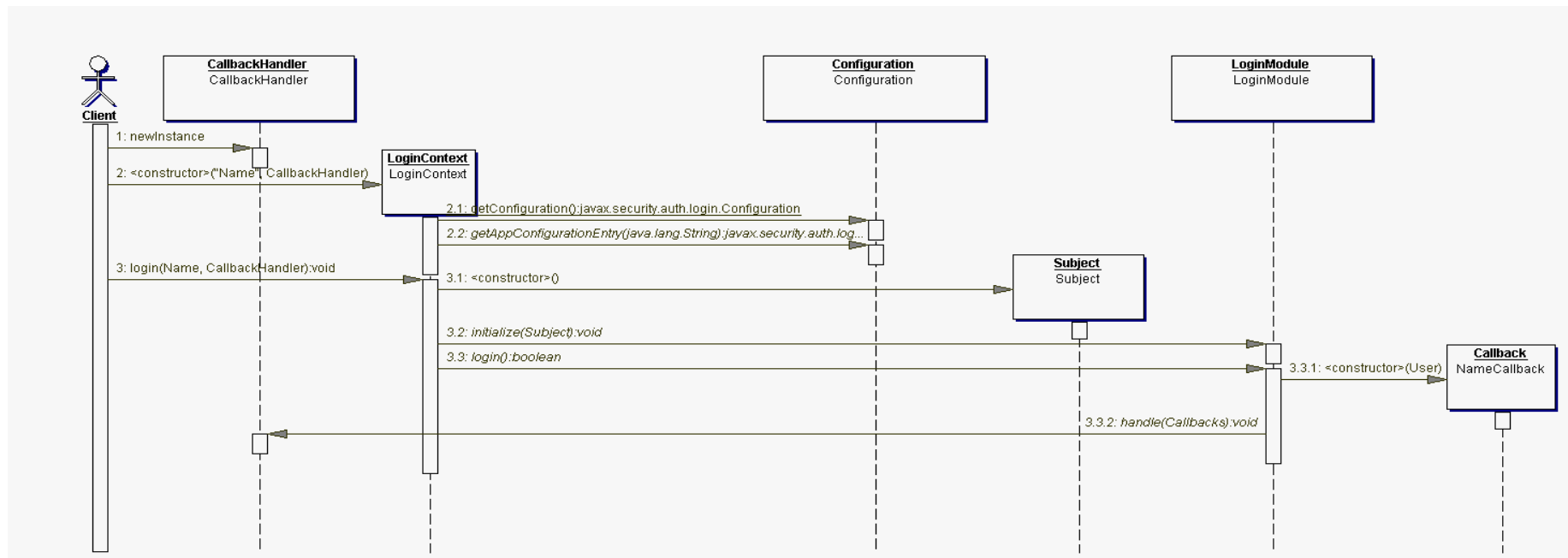
```
SimpleLogin{
    com.hotspots.javax.security.login.spi.SimpleLoginModule debug=true;
};
SSLLogin{
    com.hotspots.javax.security.login.spi.SSLLoginModule debug=true;
    com.hotspots.javax.security.login.spi.CertificateLoginModule debug=true
                                                                    url=ldap://hotspots.com;
};
default{
    com.hotspots.javax.security.login.spi.SimpleLoginModule;
};
```

Diese Parameter werden als Map der initialize-Methode des LoginModule übergeben.

```
SimpleLogin{
    com.hotspots.javax.security.login.spi.SimpleLoginModule debug=true;
};
SSLLogin{
    com.hotspots.javax.security.login.spi.SSLLoginModule requisite debug=true;
    com.hotspots.javax.security.login.spi.CertificateLoginModule required debug=true
                                                                    url=ldap://hotspots.com;
};
default{
    com.hotspots.javax.security.login.spi.SimpleLoginModule;
};
```

required	Erfolgreicher Login notwendig, die weiteren Module werden in jedem Fall abgearbeitet.
requisite	Falls der Login nicht erfolgreich ist, wird sofort abgebrochen.
sufficient	Falls der Login erfolgreich ist, werden keine weiteren Module mehr abgearbeitet.
optional	Das LoginModule kann erfolgreich sein oder auch nicht.

Der LoginContext ist eine Fassade zur Erzeugung eines Subjects



JAAS ist für den Anwendungsentwickler mit einer Ausnahme vollkommen transparent:
Die CallbackHandler-Implementierung muss bei der Instanziierung des LoginContext mit angegeben werden.

JAAS definiert nur einen globalen Default-CallbackHandler.

Eintrag unter `auth.login.defaultCallbackHandler` in der Security-Konfigurationsdatei der Java Laufzeitumgebung

Ein Authentication-Framework braucht somit nur einen CallbackHandler-Dienst
■ Ein einfaches Authentication-Framework
Also im Wesentlichen eine CallbackHandler-Factory

```
ctx = new LoginContext(DOMAIN,  
CallbackHandlerFactory.create());
```

```
try{  
    ctx.login();  
    Subject subj = ctx.getSubject();  
    log("Principals: " +  
subj.getPrincipals());  
    ctx.logout();  
}  
catch(Exception pEx){  
    ■ Verwendung des LoginContext  
    log("Failed to login: " + pEx);  
}
```

Der Provider des JAAS implementiert das `java.security.Principal`-Interface und definiert eine Klasse, die als Credential verwendet werden soll.

Der Zugriff erfolgt über das Subject

Set `getPrincipals()`

Set `getPublicCredentials()`

Set `getPrivateCredentials()`

- Prinzipal und Credential-Implementierungen

Neben dem LoginContext kann auch der aufrufende Context das assoziierte Subject liefern:

```
Subject.getSubject(java.security.AccessController.getContext())
```

Voraussetzung hierfür ist, dass die Methode direkt oder indirekt über eine der doAs()-Methoden aufgerufen wurde.

- doAs

- Die Authentifizierung erfolgt bei Enterprise JavaBeans in einem mehrstufigen Prozess:
- Der Administrator des Applikationsservers bindet diesen an ein existierendes Benutzerverwaltungssystem an.
 - Zu Testzwecken bieten Applikationsserver meistens auch ein integriertes System an. Dieses übernimmt die eigentliche Authentifizierung (wahrscheinlich mit einer Kombination aus Benutzername und Passwort).
- Weiterhin ist das Benutzerverwaltungssystem in der Lage, jedem Benutzer eine Reihe von Rollen zuzuweisen.
- Damit definiert der Administrator ein so genanntes "Security Realm".
- Jede Enterprise JavaBean kann einem Realm zugeordnet werden.
- Jede Methode kann eine der folgenden Annotations aufweisen:
 - `javax.annotation.security.RolesAllowed`, eine Liste von Rollen, denen die Ausführung erlaubt wird.
 - `javax.annotation.security.PermitAll`, die Ausführung wird jedem erlaubt. Es ist dann keine Authentifizierung notwendig.
 - `javax.annotation.security.DenyAll`, die Ausführung ist im J EE-Container verboten.
- Der Client authentifiziert sich gegen den Applikationsserver dann durch ein von diesem vorgegebenen erfahren. Im Rahmen der J EE-Spezifikation ist JAAS verpflichtend.

- `<application-policy name = "star trek">`
- `<authentication>`
- `<login-module code = "org.jboss.security.auth.spi.UsersRolesLoginModule"`
- `flag = "required">`
- `<module-option name="usersProperties">props/startrek-users.properties</module-`
`option>`
- `<module-option name="rolesProperties">props/startrek-roles.properties</module-`
`option>`
- `<module-option name="unauthenticatedIdentity">anonymous</module-option>`
- `</login-module>`
- `</authentication>`
- `</application-policy>`

- Name=Password
 - kirk=shatner
 - spock=nimoy
 - sulu=takei
 - everyone=

- Name=(Liste von Rollen)
 - kirk=captain
 - spock=firstofficer
 - sulu=navigator

```
▪ @Stateless @Remote @Local
▪ @SecurityDomain("star trek")
▪ public class EnterpriseEjb3Bean implements Enterprise{
▪     private EnterpriseA enterpriseA = new EnterpriseA();
▪     @RolesAllowed("navigator")
▪     public void changeCourse() {
▪         enterpriseA.changeCourse();
▪     }
▪     @RolesAllowed("captain")
▪     public void engage() {
▪         enterpriseA.engage();
▪     }
▪     @RolesAllowed("firstofficer")
▪     public void scan() {
▪         enterpriseA.scan();
▪     }
▪     @PermitAll
▪     public void view() {
▪         enterpriseA.view();
▪     }
▪ }
```


- Der "captain" darf Befehle erteilen ("engage!")
- Der "firstofficer" darf den Scanner benutzen
- Der "navigator" kann den Kurs der Enterprise ändern
- Enterprise anschauen darf jeder

```
public class EnterpriseEjb3Test extends TestCase {  
    public void testCaptain(){  
        Hashtable<String, String> table = new Hashtable<String, String>();  
        table.put(Context.PROVIDER_URL, "jnp://localhost:1099");  
        table.put(Context.INITIAL_CONTEXT_FACTORY,  
JndiLoginInitialContextFactory.class.getName());  
        table.put(Context.SECURITY_PRINCIPAL, "kirk");  
        table.put(Context.SECURITY_CREDENTIALS, "shatner");  
        Enterprise enterprise = ServiceLocator.getStatelessEJB(Enterprise.class);  
        enterprise.engage();  
        try{            enterprise.scan();  
                        fail("captain not allowed to scan");  
        }  
        catch(EJBAccessException e){//OK}  
        try{            enterprise.changeCourse();  
                        fail("captain not allowed to change course");  
        }  
        catch(EJBAccessException e){//OK}  
        enterprise.view();  
    }  
}
```

- Die Authentifizierung erfolgt bei Enterprise JavaBeans in einem mehrstufigen Prozess:
- Der Administrator des Applikationsservers bindet diesen an ein existierendes Benutzerverwaltungssystem an.
 - Zu Testzwecken bieten Applikationsserver meistens auch ein integriertes System an. Dieses übernimmt die eigentliche Authentifizierung (wahrscheinlich mit einer Kombination aus Benutzername und Passwort).
- Weiterhin ist das Benutzerverwaltungssystem in der Lage, jedem Benutzer eine Reihe von Rollen zuzuweisen.
- Damit definiert der Administrator ein so genanntes "Security Realm".
- Jede Enterprise JavaBean kann einem Realm zugeordnet werden.
- Jede Methode kann eine der folgenden Annotations aufweisen:
 - `javax.annotation.security.RolesAllowed`, eine Liste von Rollen, denen die Ausführung erlaubt wird.
 - `javax.annotation.security.PermitAll`, die Ausführung wird jedem erlaubt. Es ist dann keine Authentifizierung notwendig.
 - `javax.annotation.security.DenyAll`, die Ausführung ist im J EE-Container verboten.
- Der Client authentifiziert sich gegen den Applikationsserver dann durch ein von diesem vorgegebenen erfahren. Im Rahmen der J EE-Spezifikation ist JAAS verpflichtend.

- `<application-policy name = "star trek">`
- `<authentication>`
- `<login-module code = "org.jboss.security.auth.spi.UsersRolesLoginModule"`
- `flag = "required">`
- `<module-option name="usersProperties">props/startrek-users.properties</module-option>`
- `<module-option name="rolesProperties">props/startrek-roles.properties</module-option>`
- `<module-option name="unauthenticatedIdentity">anonymous</module-option>`
- `</login-module>`
- `</authentication>`
- `</application-policy>`

- Name=Password
 - kirk=shatner
 - spock=nimoy
 - sulu=takei
 - everyone=

- Name=(Liste von Rollen)
 - kirk=captain
 - spock=firstofficer
 - sulu=navigator

```
▪ @Stateless @Remote @Local
▪ @SecurityDomain("star trek")
▪ public class EnterpriseEjb3Bean implements Enterprise{
▪     private EnterpriseA enterpriseA = new EnterpriseA();
▪     @RolesAllowed("navigator")
▪     public void changeCourse() {
▪         enterpriseA.changeCourse();
▪     }
▪     @RolesAllowed("captain")
▪     public void engage() {
▪         enterpriseA.engage();
▪     }
▪     @RolesAllowed("firstofficer")
▪     public void scan() {
▪         enterpriseA.scan();
▪     }
▪     @PermitAll
▪     public void view() {
▪         enterpriseA.view();
▪     }
▪ }
```

- Der "captain" darf Befehle erteilen ("engage!")
- Der "firstofficer" darf den Scanner benutzen
- Der "navigator" kann den Kurs der Enterprise ändern
- Enterprise anschauen darf jeder

```
public class EnterpriseEjb3Test extends TestCase {  
    public void testCaptain(){  
        Hashtable<String, String> table = new Hashtable<String, String>();  
        table.put(Context.PROVIDER_URL, "jnp://localhost:1099");  
        table.put(Context.INITIAL_CONTEXT_FACTORY,  
JndiLoginInitialContextFactory.class.getName());  
        table.put(Context.SECURITY_PRINCIPAL, "kirk");  
        table.put(Context.SECURITY_CREDENTIALS, "shatner");  
        Enterprise enterprise = ServiceLocator.getStatelessEJB(Enterprise.class);  
        enterprise.engage();  
        try{            enterprise.scan();  
                        fail("captain not allowed to scan");  
        }  
        catch(EJBAccessException e){//OK}  
        try{            enterprise.changeCourse();  
                        fail("captain not allowed to change course");  
        }  
        catch(EJBAccessException e){//OK}  
        enterprise.view();  
    }  
}
```


- Wenn nun Anwendungen beispielsweise via Enterprise JavaBeans zu einer Gesamtanwendung gekoppelt werden müssen, stellen sich sofort einige Sicherheits-relevante Fragen:
 - Wie kann verhindert werden, dass bei einem JNDI-Listing die Namen und damit wohl auch die Aufgaben aller (!) vorhandenen EJBs ausgelesen werden können?
 - Wie kann garantiert werden, dass wirklich alle EJBs eine korrekte Security-Konfiguration bekommen?
 - Wie können allgemeine Plausibilitätsprüfungen sowie ein sicheres Exception-Handling an zentraler Stelle eingeführt werden?
- Die Antwort darauf ist das Design Pattern „Facade“, das in diesem Zusammenhang korrekt als „Security Facade“ benannt wird.

- Die Grundidee ist: Es gibt für den remote Zugriff nur eine einzige Enterprise JavaBean! Diese delegiert weiter und zwar:
 - Direkt an das Business Objekt.
 - Dann kann das Business Objekt aber keine eigene Security- (und/oder Transaktions-) Konfiguration bekommen.
 - An eine EJB mit lokaler Schnittstelle.
 - Dann ist eine eigene Security- (und/oder Transaktions-) Konfiguration möglich.
 - Allerdings müssen alle Beans auf dem Applikationsserver der Facade installiert sein.
 - An eine andere EJB mit remote Schnittstelle.
 - Dann ist eine eigene Security- (und/oder Transaktions-) Konfiguration möglich.
 - Die Beans müssen nicht auf dem Applikationsserver der Facade installiert sein.
 - Der Applikationsserver-Cluster mit den installierten fachlichen Beans ist netzwerktechnisch Sinnvollerweise nicht von den Clients aus sichtbar, sonst wäre die Einführung der Security Facade zumindest teilweise überflüssig.
- Die Security Facade und deren Client-Pendant, der Security Adapter, können elegant in einige simple Utilities ausgelagert werden.