HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

Sicherheit in unternehmenskritischen Applikationen

Michael Leidner

24.06.2016

Agenda

- Grundlagen
  - Funktionen von Sicherheitsmechanismen
  - Eigenschaften der Java EE Anwendungssicherheit
  - Securing Containers
  - Java EE Sicherheitsmechanismen
  - Secure Sockets Layer
  - Java SE Sicherheitsmechanismen
  - Securing GlassFish Server
  - Users, Groups, Roles und Realms
- Webanwendungen
  - Deklarative Sicherheit
  - Programmatische Sicherheit
- Enterprise Beans
  - Deklarative Sicherheit
  - Programmatische Sicherheit
- Weitere Sicherheitsmaßnahmen

Funktion von Sicherheitsmechanismen

- Schutz vor unbefugtem Zugriff auf Anwendungsfunktionen und Daten
- Mechanismen für die Nachvollziehbarkeit
- Schutz vor Ausfällen

- Sicherheitsmechanismen
  - Einfach zu verwalten
  - Transparent für den User
  - Interoperabel

Eigenschaften der Java EE Sicherheit

- Authentifizierung
- Autorisierung / Zugriffskontrolle
- Datenintegrität
- Vertraulichkeit / Datenschutz
- Unbestreitbarkeit
- Servicequalität
- Auditierung

Securing Containers

- Anwendungen bestehen aus einzelnen Komponenten
- Komponenten liegen in Containern
- Sicherheit wird von den Containern bereitgestellt

- Programmatische Sicherheit:
  - direkte Einbettung von Sicherheitsmaßnahmen in den Programmcode

- Deklarative Sicherheit:
  - Deployment Descriptor
  - Annotations

Java EE Sicherheitsmechanismen

- Application-Layer Security
  - Container sind verantwortlich
  - Schützt den Kommunikationsstorm und die Anwendungsressourcen
  - Auf die Anwendung und deren Bedürfnisse zugeschnitten
  - Sicherheitsattribute können nicht übertragen werden
  - Daten nur innerhalb der Anwendung geschützt

Java EE Sicherheitsmechanismen

- Transport-Layer Security
  - Schutz der Daten zwischen Client und Provider
  - Kommunikation mittels HTTPS mit SSL
  - Für die Authentifizierung und Nachrichtenintegrität
  - Eindeutiger Schlüssel als kryptografisches Mittel
  - Standardisierte Technologie
  - Kennt den Inhalt der Nachricht nicht
  - Keine Ende zu Ende Lösung

Java EE Sicherheitsmechanismen

- Message-Layer Security
  - Nur Empfänger kann die Nachricht entschlüsseln
  - Einzelne Bestandteile der Nachricht verschlüsseln
  - Sicherheit über alle Zwischenknoten hinweg
  - Unabhängig von der Anwendungsumgebung und dem Transportprotokell
  - Komplexer Mechanismus
  - Aufwendiger Prozess

Secure Sockets Layer

- Server identifiziert sich beim Client mit einem Zertifikat
- Server kann Zertifikat von Client verlangen
- SSL verhindert das Daten beim übermitteln geändert werden
- SSL-Antworten werden verschlüsselt
- Verschlüsseln und entschlüsseln verbraucht viel Rechenleistung

Java SE Sicherheitsmechanismen

- Java Authentication and Authorization (JAAS)
  - Erweiterbares Framework für die programmatische Sicherheit
- Java Generic Security Services (Java GSS)
  - API für den austausch von Nachrichten zwischen Anwendungen
- Java Cryptography Extension (JCE)
  - Framework für die Implementierung für die Verschlüsselung, etc.
- Java Secure Sockets Extension (JSSE)
  - Enthält eine Java-Version von SSL und TLS-Protokollen
- Simple Authentication and Security Layer (SASL)
  - Protokoll für die Authentifizierung

Securing GlassFish Server

- Open-Source-Anwendungsserver
- Unterstützt das JavaEE Sicherheitsmodell
- Untersützt verteilte Komponenten
- Konfigurationsmöglichkeiten für folgende Zwecke:
  - Hinzufügen, Löschen und Ändern von Benutzern oder Realms
  - Sicheren HTTP Listeners
  - Internet Inter-ORB Protocol (IIOP) Listeners
  - Java Management Extensions (JMX)
  - Definieren von Interfaces für die Autorisierung mittels JACC
  - Anpassungen an Authentifizierungsmechanismen
  - Einstellungen an den Richtlinienberechtigungen vornehmen

Users, Groups, Roles und Realms

- User
  - Einzelnes Individuum oder ein Anwendungsprogramm
  - Eindeutig identifizierbar
- Groups
  - User können in Groups zusammengefasst werden
  - User in einer Group haben gleiche Merkmale
  - Leichter und schneller zu verwalten als einzelne User
- Roles
  - Organisation von Zugriffsberechtigungen
  - User und Groups können Roles zugeordnet werden
- Realms
  - Ressourcen auf dem Server in Bereiche zusammenfassen
  - Eigenes Authentifizierungsschema und eigene Autorisierungsdatenbank

Deklarative Sicherheit - Sicherheitseinschränkungen

- Bei Servlets können Annotations genutzt werden.

- @ServletSecurity
- @HttpConstraint
- @HttpMethodConstraint

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

Deklarative Sicherheit - Sicherheitseinschränkungen

- <web-resource-collection> schränkt den Zugriff auf URL Bereiche ein
  - <url-pattern> enthält den eingeschränkten Bereich
  - <http-mehtod> und <http-method-omission> enthält HTTP-Methoden die geschützt oder nicht geschützt werden sollen
- <auth-constraint> enthält die Rollen die Zugriff erhalten
  - <role-name> enthält eine Rolle
- <user-data-constraint> gibt an wie die Daten transportiert werden
  - <transport-guarantee> kann den Wert CONFIDENTIAL, INTEGRAL oder NONE enthalten

Deklarative Sicherheit - Sicherheitseinschränkungen

```
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>
```

Deklarative Sicherheit - Authentifizierungsmechanismus

- Basic-Mechanismus
  - Server verlangt Name und Passwort vom Client
  - Daten werden mit einer Datenbank verglichen
  - Bei Erfolg bekommt der User Zugriff auf die geschützten Ressourcen
  - Ist der Defaultwert

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

Deklarative Sicherheit - Authentifizierungsmechanismus

- Form-based-Mechanismus
  - Eine Log-in-Seite und eine Fehlerseite kann angezeigt werden

```
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>
```

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

Deklarative Sicherheit - Authentifizierungsmechanismus

- Digest-Authentifizierung
  - Funktion wie bei der Basic-Authentifizierung
  - Client versendet Passwort als Hash

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

- Clientauthentifizierung
  - Nutzt Public-Key-Zertifikat vom Client

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

Deklarative Sicherheit - Authentifizierungsmechanismus

- Mutual-Mechanismus
  - Client und Server authentifizieren sich gegenseitig
  - Entweder mit Zertifikat oder mit Name und Passwort
  - Ablauf:
    - Client stellt Anfrage
    - Server schickt Zertifikat zur Überprüfung
    - Bei Erfolg schickt der Client sein Zertifikat zur Überprüfung

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

Deklarative Sicherheit – Sicherheitsrollen deklarieren

- Auflisten der Rollen im Deployment Descriptor

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

Programmatische Sicherheit

- Interface: HttpServletRequest
- Methoden:
  - authenticate
  - login
  - logout
  - getRemoteUser
  - isUserInRole
  - getUserPrincipal

Programmatische Sicherheit

```
1   public class LoginServlet extends HttpServlet {
2       protected void processRequest(HttpServletRequest request,
3       HttpServletResponse response)
4       throws ServletException, IOException {
5           try {
6                   request.login(userName, password);
7                   out.println(request.isUserInRole("admin")+"<br>");
8                   out.println(request.getRemoteUser()+"<br>");
9                   out.println(request.getUserPrincipal()+"<br>");
10          } finally {
11              request.logout();
12          }
13      }
14  }
```

Deklarative Sicherheit bei Enterprise Beans

- Annotations können für eine Klasse oder eine Methode angegeben werden
- Berechtigungen der Methode überschreiben die der Klasse
- Annotations:
  - @DeclareRoles: Deklarieren der Rollen, die die Klasse verwendet.
  - @RolesAllowed: Enthält die Rollen, die die Klasse oder Methode verwenden darf.
  - @PermitAll: Alle Rollen dürfen die Klasse oder Methode verwende.
  - @DenyAll: Keine Rolle darf die Methode verwenden.

Deklarative Sicherheit bei Enterprise Beans

```
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  }
```

Programmatische Sicherheit bei Enterprise Beans

- Interface: javax.ejb.EJBContext
- Methoden:
  - isCallerInRole
    - Erwartet eine Sicherheitsrolle
    - Gibt ein boolean-Wert zurück
  - getCallerPrincipal
    - Liefert das principal-Objekt zurück das den Aufrufer identifiziert

Programmatische Sicherheit bei Enterprise Beans

```
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  }
```

Weitere Sicherheitsmaßnahmen

- Datenvalidierung: Whitelisting vor Blacklisting
- Input-Validierung vom Server prüfen lassen
- SQL-Injection verhindern durch Prepared Statements
- URL-Weiterleitungen kontrollieren
- POST-Requests den GET-Requests vorziehen
- Sichere Passwörter verlangen
- Verhindern von automatisierten Angriffen

Fazit