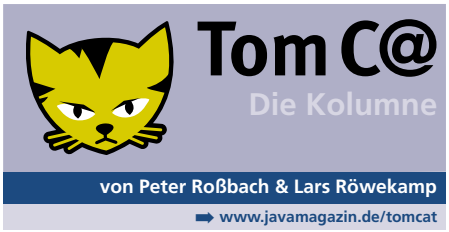
Tomcat 4 Security Realms

Ein Königreich für Tomcat

Die Möglichkeit, eine Web-Anwendung und deren Ressourcen durch Authentifizierung vor unbefugten Zugriff zu schützen, spielt im Internet eine entscheidende Rolle. Neben der klassischen, rein programmatischen Lösung, innerhalb derer sich die Sicherheitsrestriktionen im Quellcode der Anwendungen wieder finden, bietet Tomcat 4 zusätzlich ein Container Managed Security-Konzept auf Basis so genannter *Realms* an, welches wir hier vorstellen.



Die Frage der Zugriffssicherheit von Anwendungen und deren Ressourcen ist seit jeher ein essenzielles und wichtiges Thema. Dies gilt insbesondere dann, wenn es sich um Systeme handelt, welche via Inter- oder Intranet erreichbar sind. Seit der Version 3.2 des Tomcat-Servers wartet dieser mit einem Mechanismus zur Container-gestützten Sicherheit auf, welcher – auf der Grundlage von Realms – in der Lage ist, ganze Web-Anwendungen oder einzelne Ressourcen deklarativ vor unbefugtem Zugriff zu schützen.

Thema des Monats
Realms

Der aktuelle Tomcat-Server 4.1.x bietet verschiedene Mechanismen zur Container-gestützten Zugriffskontrolle auf Web-Anwendungen und deren Ressourcen an. Durch diesen Ansatz kann die Überprüfung der Zugriffsberechtigung aus der Entwicklung in das Deployment verschoben werden, was im Übrigen eine grundlegende Anforderung der Java-Servlet-Spezifikation darstellt. Dies wiederum bringt den Vorteil mit sich, dass der Entwickler einer gesicherten Anwendung kein Wissen über die endgültige Ablaufumgebung dieser besitzen muss, da die Anbindung an ein System zur Authentifizierung nun durch den Administrator vollzogen werden kann. Die Java-Servlet-Spezifikation sieht für die Authentifizierung eines Nutzers gegenüber einer Anwendung und derer Ressourcen vier verschiedene Mechanismen vor:

- BASIC
- DIGEST
- FORM
- CLIENT-CERT

Im Falle der BASIC Authentification können Login und Passwort in einem Browser-Dialog eingegeben und mit jeder HTTP-Anfrage übertragen werden. DIGESTED ist vergleichbar mit der BASIC Authentification, nur dass die Übertragung des Passworts hier nicht im Klartext, sondern SHA-, MD2- oder MD5-verschlüsselt erfolgt. FORM erlaubt die Abfrage der Authentifizierungsdaten durch ein so genanntes Customer Form, also durch eine eigene Web-Seite. CLIENT-CERT unterstützt, wie es der Name schon nahe legt, ein SSL Client-Zertifikat zur Authentifizierung.

Die Basis der Authentifizierung bildet bei allen vier Mechanismen eine Datenquelle, innerhalb derer sich die zur Authentifizierung notwendigen Daten der Nutzer sowie deren Rollenzuordnung befinden. Während sich also bei den ersten drei Verfahren mindesten der Nutzername, das Passwort sowie die zugehörigen Rollen innerhalb der Datenquelle befinden müssen, genügt bei CLIENT-CERT – dank des Zertifikats – der Nutzername und die Rolle.

Es stellt sich nun die Frage, wie genau Authentifizierung und Security-Realms zusammenhängen und wie sich deren mögliches Zusammenspiel konfigurieren lässt. Realms können als Bindeglied zwischen der Authentifizierung und den dafür zu verwendenden Datenquellen angesehen werden, indem sie von der zugrunde liegenden Datenquelle abstrahieren und deren Zugriffsmechanismen kapselt. Etwas einfacher ausgedrückt bedeutet dies, dass die eigentliche Authentifizierung auf Security-Realms basiert, welche für sie wie eine Datenquelle fungieren. Denkbare Datenquellen sind dabei zum Beispiel LDAP-Server, RDBMS- oder aber XML-basierte Dateien. Es ist an dieser Stelle wichtig zu bemerken, dass nicht jeder Authentifizierungsmechanismus auch mit jedem der zur Verfügung stehenden Realms wirklich Sinn macht bzw. überhaupt möglich ist. Die Verwendung von Realms ermöglicht zwar generell eine Authentifizierung, ist häufig aber nur in Kombination mit anderen Sicherheitsmaßnahmen innerhalb der Konfiguration des Servers oder der Web-Applikation wirklich sicher und somit sinnvoll. So sollte zum Beispiel bei einer geschützten Ressource auch die eigentliche Übertragung mittels *Transport Guarantee* vor unbefugtem Zugriff gesichert sein [1], [2].

Kommen wir nun zur Konfiguration einer sicheren Anwendung. Möchte man eine Web-Anwendung vor unbefugtem

Listing 1: Security Constraint innerhalb der Konfigurationsdatei *web.xml*

```
<web-app>
 ...
 <security-constraint>

  <web-ressource-collection>
   <web-ressource-name>
    Secure Web Application
   </web-ressource-name>
   <url-pattern>/secure/*</url-pattern>
  </web-ressource-collection>

  <auth-constraint>
   <role-name>
    admin
   </role-name>
  </auth-constraint>

 </security-constraint>

 <user-data-constraint>
  <transport-guarantee>
   CONFIDENTIAL
  </transport-guarantee>
 </user-data-constraint>
 ...
</web-app>
```

Listing 2: Authentifizierungsmechanismus innerhalb der Konfigurationsdatei *web.xml*

```
<web-app>
 ...
 <security-constraint>
  ...
 </security-constraint>

 <login-config>
  <auth-method>BASIC<auth-method>
  <realm-name>Secure Web Application</realm-name>
 </login-config>
 ...
</web-app>
```

```
<Server ...>
 ...
 <Engine ...>
  ...
  <Realm className="org.apache.catalina.realm.
                                    UserDatabaseRealm"
     debug="0"
     resourceName="UserDatabase"/>
  ...
 </Engine>
 ...
</Server>
```

Zugriff schützen, sind dazu zunächst zwei Schritte und damit zwei neue Einträge in der Konfigurationsdatei *web.xml* der Web-Anwendung notwendig:

- Schritt 1: Bestimmen der zu schützenden Ressourcen.
- Schritt 2: Definieren des zu nutzenden Authentifizierungsmechanismus.

Zur Ressourcenbestimmung muss eine *Security Constraint* innerhalb der Konfigurationsdatei *web.xml* deklariert werden. Listing 1 zeigt exemplarisch eine Security Constraint aus der *web.xml* der Beispielanwendung */basicsecure*, deren Sourcen sich auf der Heft-CD befinden.

Die entscheidenden Tags innerhalb der Security Constraint sind zum einen *<url-pattern>* zur Angabe der zu schützenden Ressourcen und zum anderen *<role-name>*, mit dessen Hilfe diejenige(n) Rolle(n) bestimmt werden, welchen der Zugriff erlaubt werden soll. Durch die Angabe des *User Data Constraint* bzw. der darin enthaltenen *Transport Guarantee* wird zusätzlich gewährleistet, dass Daten während der Übertragung nicht verändert oder von einem Dritten gelesen werden. Die *User Data Constraint* ist an dieser Stelle zwar nicht unbedingt notwendig aber für die Übertragung von Login-Informationen durchaus sinnvoll.

Das Definieren des Authentifizierungsmechanismus wird durch folgenden Eintrag innerhalb der Konfigurationsdatei erreicht, der als Authentifizierungsmethode *BASIC Authentification* auswählt (Listing 2).

Durch die beiden bisher ausgeführten Schritte haben wir die Web-Anwendung theoretisch bereits gegen unbefugten Zugriff gesichert. Was wir allerdings noch nicht erreicht haben, ist eine Verbindung zu einer Datenquelle, in der sich die Nutzer- und Rolleninformationen befinden. Genau hier kommen die Realms und deren Konfiguration ins Spiel. Das zu verwendende Security-Realm – und damit die Verbindung zu einer Authentifizierungs-Datenquelle – kann innerhalb der Server-Konfiguration *server.xml* durch Hinzufügen des Tags *<Realm classname="[Klassenname des Realms]" ... >* angegeben werden [3], [4]. Listing 3 zeigt den notwendigen Eintrag für ein Realm vom Typ *UserDatabase* innerhalb des *<Engine>*-Tags der Server-Konfiguration.

Das *<Realm>*-Tag kann sowohl im Element *<Engine>* als auch in den Elementen *<Host>* und *<Context>* vorkommen. Ist das Realm als direktes Unterelement der Engine deklariert, so bezieht es sich auf alle virtuellen Hosts der Engine und alle damit verbundenen Kontexte. Findet die Deklaration dagegen innerhalb eines *<Host>*-Elements statt, wird das Realm lediglich vom virtuellen Host selbst genutzt. Die stärkste Einschränkung ergibt sich bei einer Deklaration innerhalb eines Kontext-Elements, da in diesem Fall nur der umgebende Kontext das Realm benutzt.

Standard-Realms

Das mit Abstand einfachste, gleichzeitig aber auch am wenigsten flexible Realm ist das Memory-Realm, welches durch die Klasse *org.apache.catalina.realm.MemoryRealm* implementiert wird. Mit seiner Hilfe können Nutzerinformationen zum Zeitpunkt des Serverstarts aus einer Konfigurationsdatei gelesen und zur Authentifizierung innerhalb des Lebenszyklus der Web-Anwendung genutzt werden.

Die Basis für dieses Realm stellt eine XML-Datei dar, welche standardmäßig unter dem Namen *tomcat-users.xml* im Verzeichnis *$CATALINA_HOME$/conf* zu finden ist. Alternativ kann durch die Angabe des Attributs *pathname* innerhalb des *<Realm>*-Tags jede beliebige andere

XML-Datei genutzt werden, die dem Aufbau der Datei *tomcat-users.xml* entspricht. Listing 4 zeigt beispielhaft eine einfache *tomcat-users.xml*.

So einfach ein Memory-Realm zu nutzen ist, so begrenzt ist auch seine Flexibilität. Bedingt durch die Tatsache, dass die Nutzerinformationen aus einer Konfigurationsdatei gelesen werden, handelt es sich bei diesem Mechanismus um einen read-only Ansatz. Änderungen an der XML-basierten Konfigurationsdatei wirken sich erst nach einem Neustart des Servers aus, was die praktische Verwendbarkeit der Memory-Realms auf statische Deployments beschränkt.

Deutlich mächtiger als das eben dargestellte Memory-Realm zeigt sich das JDBC- Realm. Im Gegensatz zum Memory-Realm führt es die Authentifizierung nicht gegen Nutzerinformationen innerhalb einer Datei, sondern gegen Tabellen einer JDBC-basierten Datenbank durch. Dies bringt zum einen eine deutlich höhere Flexibilität mit sich und erlaubt zum anderen die Modifikation der Authentifizierungsdaten zur Laufzeit.

Das JDBC-Realm geht davon aus, dass zwei Tabellen namens *users* und *user_roles* existieren. Während die Tabelle *users* die Namen der Nutzer sowie ihre Passwörter beinhaltet, findet sich in der Tabelle *user_roles* das Mapping zwischen den Nutzern und ihren Rollen. Entsprechend befinden sich in der Tabelle *users* – als minimale Voraussetzung – die Spalten *user_name* und *user_pass* und in der Tabelle *user_roles* die beiden Spalten *user_name* und *user_role*. Als Primarschlüssel der Tabelle *users* dient die Spalte *user_name*. Tabelle 1 zeigt die genaue Spezifikation der Tabelle *users*, Tabelle 2 die Spezifikation der Tabelle *user_roles*.

Aufgrund der höheren Flexibilität verfügt das <*Realm*>-Tag des JDBC-Realms über wesentlich mehr Attribute [5] als die oben vorgestellte Alternative Memory-Realm. So ist es zum Beispiel möglich, mittels Tag Attribute nicht nur mit Hilfe der beiden oben beschriebenen Tabellen zu authentifizieren, sondern auch bestehende User Management-Systeme anzubinden. Obwohl die Feldtypen und -längen der beiden virtuellen Tabellen durch die Spezifikation fest vorgegeben sind, bedeutet dies nicht, dass auch die physikalischen Tabellen genau diese Restriktionen einhalten müssen. Als Feldtypen werden alle Typen akzeptiert die mittels Treiber in einem String konvertiert werden. Für die Feldlänge gilt, dass längere Felder erlaubt sind und automatisch angepasst werden.

Wie schon zuvor das JDBC-Realm, ermöglicht auch das JNDI-Realm die Anbindung eines bereits vorhandenen externen Systems zur Authentifizierung. Nur dass es sich in diesem Fall nicht um eine Datenbank, sondern um einen LDAP Directory Service handelt, auf den mittels JNDI-Provider zugegriffen werden kann. Um einen Directory Service sinnvoll zur Authentifizierung und somit zum Zugriffsschutz für Web-Anwendungen oder deren Ressourcen nutzen zu können, müssen dem Realm entsprechende Abbildungsinformationen mit auf den Weg gegeben werden. Neben allgemeinen Verbindungs- informationen werden vor allem Angaben zum Auffinden der Nutzer- und Rolleninformationen benötigt, welche über ein „Pattern Matching“ die notwendigen Informationen aus den LDAP-Knoten auffinden und extrahieren. JNDI-Realms sind sehr flexibel und können daher mit verschiedenen LDAP-Strukturen interagieren. Ihre Flexibilität ist aber gleichzeitig auch einer ihrer größten Nachteile, da sich diese auch in der Komplexität der Konfiguration widerspiegelt. Wirft man einmal einen Blick in die bekannte Tomcat- Newsgroup [6], so fällt auf, dass es gerade bei der Anwendung dieses Realm-Typs immer wieder zu erheblichen Problemen kommt.

Listing 4: *tomcat-users.xml* mit *jmColumnReader* und *superUser*-Rolle

```
<tomcat-users>
 <role rolename="tomcat"/>
 <role rolename="admin"/>
 <role rolename="manager"/>
 <role rolename="jmColumnReader"/>
 <user name="tomcat"
    password="tomcat"
    roles="tomcat" />
 <user name="admin"
    password="admin"
    roles="admin" />
 <user name="manager"
    password="manager"
    roles="manager" />
 <user name="jmColumnReader"
    password="jmColumnReader"
    roles="jmColumnReader" />
 <user name="superUser"
    password="superUser"
    roles="admin,manager,tomcat,jmColumnReader" />
</tomcat-users>
```

Weitere Realm-Typen

Neben den bisher angesprochenen Realm-Arten gibt es in der Version 4 des Tomcat noch das *DataSource-Realm*, das *JAAS-Realm* und das *UserDatabase-Realm*. Das DataSource Realm kann in Aufbau und Funktion mit dem JDBC-Realm verglichen werden kann. Der wesentliche Unterschied besteht in der Art des Zugriffs: Während das JDBC-Realm direkt via JDBC-Treiber auf die zugrundeliegende Datenbank zugreift, nutzt das DataSource-Realm den JNDI-Kontext.

Das JAAS-Realm nutzt zur Authentifizierung den *Java Authentication and Authorization* Service und benötigt daher entweder das JDK 1.4 oder aber das *jaas.jar*-Archiv als Plugin für das JDK 1.3.

Das UserDatabase-Realm schließlich ist vergleichbar mit dem Memory-Realm, nur das es dessen entscheidenden Nachteil – den read-only Ansatz – umgeht und persitente Änderungen an den Nutzerinformationen und somit an der XML-basierten Datei erlaubt.

User Defined Realms

Auch wenn die im Tomcat bereits enthaltenen Realms sicherlich für einen Großteil an denkbaren Anwendungsszenarien ausreichen, ergibt sich die extreme Flexibilität erst durch die Möglichkeit, eigene User Defined Realms zu implementieren und somit völlig losgelöste Authentifizie-

Spaltenname	Feldtyp	Feldlänge
user_name	VARCHAR NOT NULL	15
user_pass	VARCHAR NOT NULL	15

Tab. 1: Aufbau der Tabelle *users*

Spaltenname	Feldtyp	Feldlänge
user_name	VARCHAR NOT NULL	15
user_role	VARCHAR NOT NULL	15

Tab. 2: Aufbau der Tabelle *user_roles*

rungsmechanismen anzubieten. Alles was es dazu bedarf, ist eine Klasse welche die Schnittstelle *org.apache.catalina.realm.Realm* implementiert. Um sich den Aufwand für die Umsetzung aller dort deklarierten Methoden zu sparen, ist es ratsam auf die Standardimplementierung *org.apache.catalina.realm.RealmBase* zurückzugreifen und von dieser abzuleiten. Wählt man diesen Weg, müssen lediglich diejenigen Methoden überschrieben werden, die von der Standardimplementierung abweichen, also in der Regel die Methoden, innerhalb derer die eigentliche Authentifizierung stattfindet.

Profi Know-how
Dynamische Sicherheit mit UserDatabase-Realms

Wie wir im bisherigen Verlauf dieser Kolumne gesehen haben, hat jedes der vorgestellten Realms seine Vor- und Nachteile. Das Memory-Realm ist leicht anzuwenden, dafür aber wenig flexibel. Ein zusätzlicher Nachteil ergibt sich aus der Tatsache heraus, dass eine Änderung an den Authentifizierungsdaten einen Server-Neustart bedingt, was in einer produktiven Umgebung sicher nicht anwendbar ist.

Die JDBC- und JNDI-Realms sind in der Lage, bestehende Systeme anzubinden und bieten gleichzeitig einen hohe Flexi-

bilität. Alle Modifikationen an den Authentifizierungsdaten werden ohne Server-Neustart zur Zugriffskontrolle angewendet. Gerade wenn mehrere Tomcat-Server zu einem Cluster zusammengefasst sind, bietet sich die Nutzung der externen Datenquelle an. Ein Nachteil ergibt sich aus ihrer Komplexität. Darüber hinaus muss ein externes System, wie zum Bespiel ein RDBMS oder ein LDAP-Server vorhanden sein.

Doch was ist, wenn man einerseits einen flexiblen Ansatz wünscht, aber anderseits noch kein bestehendes User Management System zur Anbindung vorliegt? Muss in einem solchen Fall extra ein LDAP-Server oder eine Datenbank aufgesetzt und entsprechend der Realm-Vorgaben konfiguriert und eingerichtet werden? Dieses Problem hat auch die Tomcat Group erkannt und mit Hilfe des UserDatabase-Realms, welcher seit der Version 4.1 zur Verfügung steht, Abhilfe geschaffen.

Das UserDatabase-Realm kann am besten als ein Memory-Realm beschrieben werden, welches zusätzlich über Schreibrechte verfügt. Durch diesen Ansatz wird automatisch der größte Nachteil des Memory-Realms umgangen – das read-only-Problem. Als Eingabe dient dem UserDatabase-Realm eine XML-Datei, welche dem Aufbau der *tomcat-users.xml* gleicht. Während des gesamten Lifcycles des Realms können die Nutzerinformationen bearbeitet und persistent gemacht werden, wobei die Änderungen zurück in die XML-Datei geschrieben werden. Ein Vorteil gegenüber den deutlich komplexeren JDBC- und JNDI-Realm-Lösungen ist das einfache Mapping zwischen Nutzer- und Rolleninformationen sowie die Tatsache, dass die Administration der Nutzer über die in Tomcat enthaltene Administrator-Anwendung vollzogen werden kann, da diese ebenfalls auf die UserDatabase zurückgreift (Abb. 1).

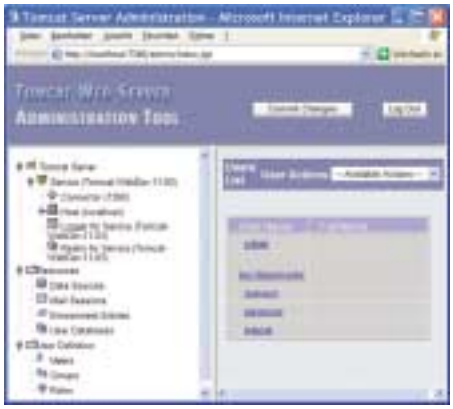


Abb. 1: Zugriffsrechte via Administrator-Applikation

Das folgende Beispiel soll die Anwendung eines UserDatabase-Realms verdeutlichen. Sämtliche Quellcodes der Beispielanwendung */basicsecure* sowie einer zwei- ten Beispielanwendung */formsecure* finden sich auf der Heft-CD. Die Anwendung *basicsecure* stellt eine kleine administrative Oberfläche zur Verwaltung von Nutzerinformationen dar. Natürlich haben nur Nutzer mit der Rolle Administrator das Recht, auf die geschützten Bereiche der Anwendung und deren Ressourcen zuzugreifen. Die Authentifizierung in wird mittels BASIC Authentication vollzogen (Abb. 2). Eine Alternative hierzu wäre die in Abbildung 3 dargestellte FORM-basierte Authentifizierung, welche im Beispiel */formsecure* zur Anwendung kommt.

In unserem Beispiel soll die UserDatabase als JNDI-Ressource des Servers konfiguriert werden, um so später innerhalb der Anwendung via JNDI auf sie zugreifen

Listing 5: Konfiguration der UserDatabase als JNDI-Ressource

```
<Server ...>
 ...

<GlobalNamingResources>
 ...
 <Resource name="UserDatabase" auth="Container"
     type="org.apache.catalina.UserDatabase"
  description="Modifizierbare Nutzer-/
                                  Rechteverwaltung">
 </Resource>

 <ResourceParams name="UserDatabase">
  <parameter>
   <name>factory</name>
   <value>org.apache.catalina.users.
                 MemoryUserDatabaseFactory</value>
  </parameter>
  <parameter>
   <name>pathname</name>
   <value>conf/tomcat-users.xml</value>
  </parameter>
 </ResourceParams>

</GlobalNamingResources>

<Engine ...>
 <Host...>...
  <Context path="/basicsecure" privileged="true"...>
   ...
   <ResourceLink name="userDatabase" global=
                                  "UserDatabase"
      type="org.apache.catalina.UserDatabase"/>
   ...
  </Context>
 </Host>
</Engine>

...
<Server>
```

Listing 6: Zugriff auf die UserDatabase via JNDI Lookup

```
...
Context initCtx = new InitialContext();
Context envCtx = (Context)
initCtx.lookup("java:comp/env");
UserDatabase database = (UserDatabase) envCtx.
                                  lookup("userDatabase");
...
```

Listing 7: Direkter Zugriff auf die UserDatabase ohne JNDI Lookup

```
...
StandardServer server = (StandardServer)
ServerFactory.Server();
Context envCtx = server.getGlobalNamingContext();
UserDatabase database = (UserDatabase) envCtx.
                                  lookup("userDatabase");
...
```

Listing 8: UserDatabase Realm Konfiguration

```
<Server ...>
 ...
 <Engine ...>
  ...
  <Realm className="org.apache.catalina.realm.
                                  UserDatabaseRealm"
     debug="0"
     resourceName="UserDatabase"/>
  ...
 </Engine>
 ...
</Server>
```

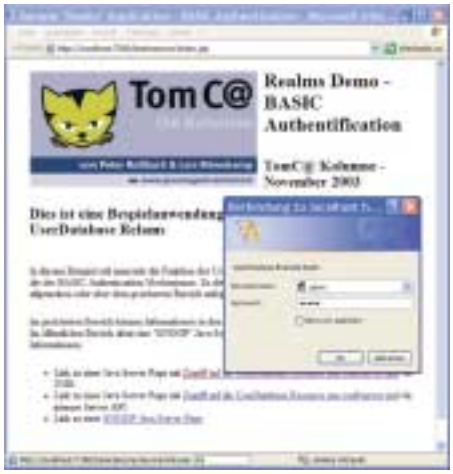


Abb. 2: Authentifizierung via BASIC Authentication

zu können. Dazu ist der Eintrag aus Listing 5 innerhalb der Server-Konfigurationsdatei *server.xml* notwendig.

Durch den *GlobalNamingRessource*-Eintrag innerhalb der *server.xml* ist die UserDatenbank mittels JNDI-Lookup erreichbar. Durch den zusätzlichen *RessourceLink*-Eintrag wird die notwendige Verbindung zwischen der globalen Ressource UserDatabank und der Anwendung */basicsecure* hergestellt, in welcher die Ressour ce über den Namen *userDatabase* angesprochen werden kann. Der Zugriff innerhalb einer Anwendung muss relativ zu dem Naming Context *java:comp/env* erfolgen (Listing 6).

Ebenfalls möglich ist ein direkter Zugriff unter Umgehung des JNDI-Kontextes, der allerdings die Verwendung der internen Tomcat APIs und die dafür notwendigen Rechte voraussetzt (Listing 7).

Natürlich soll unsere kleine Web-Anwendung nicht nur die UserDatabase als Ressource nutzen, sondern die Kontrolle des Zugriffs auf */basicsecure* selbst auch durch die Einträge innerhalb der konfigurierten UserDatabase erfolgen. Zu diesem Zweck muss ein entsprechendes <*Realm*>- Tag zu unserer *server.xml* hinzugefügt (Listing 8) und die Web-Anwendung mit den in Listing 1 bereits vorgestellten *Security Constraints* versehen werden.

Alternativ könnte man als Entwickler den Zugriff auf die UserDatabase und deren Informationen auch via JMX – also über den internen MBean-Server des Tomcat – vollziehen [7]. Dies ist exakt der Weg, den die Administrator-Anwendung des Tomcat-Servers geht, der aber gleichzeitig Know-how zu Aufbau und Verwendung des MBean-Servers voraussetzt.

Fazit

Wie sich im Verlauf der Kolumne gezeigt hat, bietet Tomcat 4 mit dem Realm-Konzept einen mächtigen Mechanismus zur Umsetzung von Container Managed Security an, welche wiederum eine Anforderung der Java-Servlet-Spezifikation 2.3 dar- stellt. Durch die verschiedenen im Tomcat 4 vorhandenen Realms wird bereits ein breites Spektrum an Authentifizierungsmechanismen abgedeckt. Es kann sowohl gegen bestehende Datenbanken (via JDBC- oder DataSource-Realm) als auch gegen LDAP-Server (via JNDI-Realm) authentifiziert werden. Soll kein externes System zur Speicherung der Nutzer- und Rolleninformationen verwendet werden, besteht die Möglichkeit, das Memory-Realm (read-only) oder aber das UserDatabase-Realm zu verwenden. Findet sich in keinem der vorgegebenen Realms die gewünschte Lösung, bietet das Realm-Interface die Möglichkeit einer Eigenimplementierung und somit der Umsetzung einer individuellen Authentifizierungsmethode.

Unabhängig davon, welcher Realm-Typ letztendlich genutzt wird, ist die Feststellung wichtig, dass durch ihre Verwendung administrative Aufgaben durchgeführt werden können ohne dabei – wie es die Administrator-Anwendung des Tomcat tut – den internen MBean-Server verwenden zu müssen. So können zum Beispiel mit Hilfe eines Realms sich innerhalb einer Web-Anwendung registrierende Nutzer direkt in die Datenquelle des Realms eingetragen werden und stehen somit ohne weiteren administrativen Aufwand direkt dem Authentifizierungsmechanismus zur Verfügung.

Als kleine Anmerkung und gleichzeitig als Warnung sei an dieser Stelle noch einmal erwähnt, dass die Verwendung von Security-Realms in Kombination mit Security Constraints allein häufig nicht ausreicht, um eine Web-Anwendung im gewünschten Grad gegen unbefugten Zu- griff zu schützen. Im Sinne des mathematischen Ausdrucks *die Bedingung*

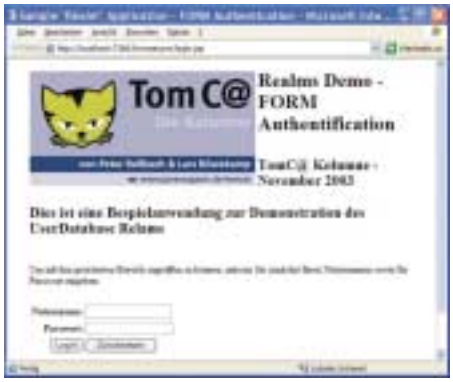


Abb. 3: Authentifizierung via FORM Authentication

ist notwendig aber nicht hinreichend sollten in den meisten Fällen noch – wie im obigen Beispiel gezeigt – ergänzende Transport-Sicherheitsmechanismen via HTTPS Connector zur Hilfe herangezogen werden.

Und beim nächsten Mal...

In der nächsten Ausgabe des *Java Magazins* werden wir uns eingehend mit der Thematik der Coyote HTTP-Connectoren befassen. Wir werden ausgiebig die Bedeutung der Connector-Parameter beschreiben und den Blick auf die Performance richten.

Wir freuen uns schon auf Kommentare und Anregungen – besuchen sie also unsere Tom C@-Site und das Tom C@-Forum oder die Tomcat Mailing-Listen [6], [7], [8]. ■

Links & Literatur

[1] Jason Hunter, William Crawford „Java Server Programmierung", O'Reilly 2002

[2] Servlet API 2.3: java.sun.com/servlets/

[3] Vivek Chopra, Ben Galbraith, Gotham Polisettym Brian P. Rickabaugh, John Turner: „Apache Tomcat Security Handbook", Wrox Press, 2003

[4] Peter Roßbach (Hrsg.); Andreas Holubek, Thomas Pöschmann, Lars Röwekamp, Peter Tabatt: „Tomcat 4X: Die neue Architektur und moderne Konzepte für Webanwendungen im Detail"; Software & Support Verlag, 2002

[5] jakarta.apache.org/tomcat/tomcat-4.1-doc/realm-howto.html

[6] www.mail-archive.com/tomcat-user@jakarta.apache.org/

www.mail-archive.com/tomcat-dev@jakarta.apache.org/

[7] Peter Roßbach, Lars Röwekamp; „Administration und Management des Tomcat entzaubert", *Java Magazin* 10.2003

[8] www.javamagazin.de/tomcat/

[9] tomcat.objektpark.org/