Tomcat als Trutzburg: Sicherheit für Leib und Leben

Mein Tomcat ist eine Festung …

Innerhalb eines Tomcat-Web-Container tummeln sich idealerweise etliche Hosts mit den verschiedensten Anwendungen. Da nicht alle „Kunden" immer nur Gutes im Schilde führen, sind entsprechende Schutzmaßnahmen dringend notwendig. Die Anwendungen und Anwender, die sich einen Rechner bzw. Server teilen, müssen voreinander geschützt werden. In dieser Kolumne wollen wir den Aufbau und die Grundlagen zur Aufrüstung einer Basisfestung – der Trutzburg – für den Tomcat diskutieren und einem Angriff unterziehen.

Es war einmal vor langer Zeit in einem weit entfernten Land … So ähnlich beginnen fast alle Märchen, die wir in der Kindheit kennen gelernt haben. Auch wir wollen hier ein kleines Märchen erzählen – allerdings eines aus der heutigen Zeit. Es handelt von Burgen, bösen Angreifern und einer nahezu aussichtslosen Schlacht. Doch wie in einem Märchen üblich siegt am Ende natürlich das Gute.

Viele Anbieter von Webanwendungen nutzen heutzutage die Gelegenheit des so genannten Hosting, bei dem die eigenen Webanwendungen auf einem externen Server der Öffentlichkeit kostengünstig zugänglich gemacht werden. Die Gründe für ein wenig mehr Sicherheit der Anwendungen, die innerhalb eines solchen Servers ablaufen, liegen auf der Hand. Die aus Kostengründen geteilten Ressourcen sollen natürlich geteilt bleiben. Wir Java-Entwickler haben da mit den Möglichkeiten unserer Programmierumgebung das große Los gezogen. Die Java Virtual Machine (JVM) bietet ausgereifte Sicherheitsmechanismen, die uns als Betreiber von Tomcat-Servern geradezu auffordert, eine gesicherte Festung hinter unserer Firewall zu bauen. Auf einem Server laufen meist verschiedene Services und die sollten in verantwortlicher und kontrollierter Weise miteinander umgehen. Der Administrator hat die Aufgabe, sowohl den Tomcat abzusichern, die notwendigen Services freizugeben als auch die einzelnen Webanwendungen entsprechend sinnvoll einzuschränken.

Wir wollen nach einer kurzen Einführung in die grundsätzlichen Tiefen des Java Security Managers die praktische Umsetzung einer Trutzburg für den Tomcat beschreiben und einem Sicherheits-Check unterziehen.

Von Bausteinen und Techniken zum Burgbau: der Java Security Manager

Aber wie sichert man den Server und Tomcat so ab, dass die für das oben geschilderte Szenario notwendigen Sicherheitsregeln eingehalten werden? Nichts einfacher als das: Für solche Aufgaben gibt es bekanntlich den Security Manager [1]. In jeder JVM ist ein solcher Security Manager enthalten. Um diese Eigenschaft zu benutzen, brauchen wir ihn nur zu aktivieren:

```
java -cp ... -Djava.security.Manager -Djava.security.policy=
                                    PFAD_ZUR_POLICYDATEI ...
```

Mit dieser Option der JVM werden alle sicherheitsrelevanten Aktionen zunächst an den Security Manager umgeleitet. Dieser vergleicht die Aktion mit den Regeln in der Policy-Datei und entscheidet dann, ob die angedachte Aktion erlaubt wird oder nicht. Die Policy-Datei kann dabei an jedem beliebigen Ort gespeichert sein. Als einzige Auflage gilt, dass sie im Aufruf angegeben werden und lesbar sein muss. Im Tomcat befinden sich die Sicherheitsregeln im Verzeichnis *conf* in der Datei *catalina.policy*.

Aufbau einer Policy-Datei

Sobald der Security Manager aktiviert ist, darf eine Java-Klasse erst einmal gar nichts mehr. Alle Aktionen wie z.B. das Lesen/Schreiben von Properties, Dateisystemzugriffe, Socket-Zugriffe sind per se verboten. Mit einer Policy-Datei geben wir nun jede notwendige und von uns gewollte Aktion mittels entsprechender Regeln wieder frei und ermöglichen so, dass die entsprechende Klasse einwandfrei arbeiten kann. Auf der Heft-DVD finden Sie eine vollständige Policy-Datei, mit der Sie Ihren Tomcat starten können. Im Folgenden erklären wir kurz an drei Beispielen den typischen Aufbau einer solchen Regel. Für detaillierte Informationen zu dem Aufbau einer Policy-Datei surfen Sie bitte auf Suns Sicherheitsdokumentation [2]. Sie sehen das im ersten Beispiel einer Sicherheitsregel.

```
grant codeBase "file:${java.home}/lib/-" {
    permission java.security.AllPermission;
};
```

Die Regel im ersten Beispiel gibt den JARs im Verzeichnis *lib* der Standard-Java-Installation alle Rechte. Diese Regel beinhaltet zwei Vereinfachungen in ihrer Schreibweise. Systemeigenschaften (*System.getProperty(String)*) können beliebig mit dem Konstrukt *${name}* benutzt werden. Bei den Angaben von Ressourcen im Dateisystem können Platzhalter verwendet werden: 1. „-“ (Bindestrich) für alle Dateien und Unterverzeichnisse rekursiv, 2. „*“ nur für alle Dateien in dem angegebenen Verzeichnis. Ein kleiner Tipp am Rande: Verwenden Sie diese Optionen so oft wie möglich. Sie sparen damit eine Menge zusätzlicher Regeln und Sie ermöglichen es dem Security Manager, etwas schneller zu arbeiten. Achten Sie aber unbedingt darauf, dass Sie nicht aus Bequemlichkeit mehr Rechte freigeben, als Sie eigentlich wollen.

```
grant {
  permission java.util.PropertyPermission "os.name", "read";
  permission java.util.PropertyPermission "os.version", "read";
}
```

Das oben stehende zweite Beispiel zeigt, wie Sie Regeln für alle Klassen freigeben. Wenn keine *codebase* konfiguriert ist, gilt die Regel für alle Programmteile. Die Anwendung dieser Regel ist typisch für die Freigabe der Permission *java.util.PropertyPermission*. Im folgenden Beispiel sehen Sie, dass auch mehrere Permissions gleichzeitig für eine *codebase* freigegeben werden können.

```
grant codeBase
 "file:${catalina.base}/webapps/examples/-" {
   permission java.net.SocketPermission "dbhost.
                              mycompany.com:5432", "connect";
   permission java.io.FilePermission "${catalina.base}/
                                 webapps/examples", "write";
}
```

Welche Permissions gibt es?

Im Folgenden werden die gebräuchlichsten Permissions im Detail beschrieben. Wenn Sie tiefer in das Thema einsteigen wollen, dann empfehlen wir Ihnen einen Blick auf [3], [4], [5].

- *java.security.AllPermission*
  Diese Permission gibt der angegebenen Ressource alle Rechte. Hier brauchen Sie keine zusätzlichen Parameter:

```
grant codebase "codebase"{
permission java.security.AllPermission;
};
```

- *java.io.FilePermission*
  Die Permission, die Sie wohl am meisten verwenden werden. Hiermit regeln Sie den kompletten Zugriff auf Dateisysteme. Sie können die Aktionen *read*, *write*, *execute* und *delete* verwenden.

```
grant codebase codebase{
permission java.io.FilePermission-"Zielpfad", "read,
                                    write, execute, delete";
};
```

- *java.net.SocketPermission*
  Mit dieser Permission geben Sie alles frei, was mit Sockets zu tun hat. Der Host, den die Anwendung kontaktieren möchte oder der die Anwendung kontaktieren möchte, wird so angegeben: *(Hostname | IP-Adresse) [: Portbereich]*.

  Dabei kann der Portbereich wie folgt angegeben werden: *portnumber | portnumber|portnumber-[portnumber]*.

  Als Aktion können Sie *accept*, *connect*, *listen* und *resolve* verwenden.

```
grant codebase codebase{
permission java.net.SocketPermission host [:port],
accept, connect, listen, resolve;
};
```

- *java.util.PropertyPermission*
  Diese Permission regelt den Lese- und Schreibzugriff auf Systemeigenschaften. Die Aktion *read* ruft die Methode *System.getProperty()* und die Aktion *write* nutzt die Methode *System.setProperty()*.

```
grant [codebase codebase]{
permission java.util.propertyPermission name, read,
write;
};
```

Bei dieser Permission können Sie auch mit „*“ als Platzhalter arbeiten: *java.**, *read* gibt alle Properties, die mit *java.* anfangen, zum Lesen frei.

Abb. 1: Der Sicherheit des Tomcat mit dem *securitycheck.war* auf den Zahn gefüllt: Autsch …

Der Bau der Tomcat-Trutzburg mithilfe des Security Managers

Nach der Beschreibung der Technik des Security Manager, zu den Policies und Sicherheitsregeln wollen wir wieder zu unserem Ausgangspunkt – dem Bau einer sicheren Trutzburg – zurückkehren und untersuchen, was wir unternehmen müssen, um konkrete Sicherheitsziele mit dem Tomcat zu erfüllen.

Grundsätzlich haben sich in unserer Praxis folgende, sicherheitsrelevante Regeln bewährt:

- Webanwendungen dürfen nur Dateien in ihrem eigenen *${docBase}* lesen.
- Webanwendungen sind ausschließlich im *${docBase}* und nicht verstreut über das Dateisystem zu finden.
- Protokolldateien (*AccessLog* und *SystemLog*) werden für jeden virtuellen Host in einem speziellen Verzeichnis *logs* geschrieben, der außerhalb des *${docBase}* liegt.
- Es dürfen nur Daten der eigenen Anwendung verändert werden.
- Temporäre Dateien werden in einen dafür vorgesehen Ordner geschrieben und gelesen.
- Systemeinstellung sollen nicht verändert werden.
- Socket-Verbindungen zu allen Rechnern der Ports 25 (SMTP), 80 (HTTP), 443 (HTTPS) und den lokalen Datenbanken werden gewährt.

Als Erstes starten wir den Tomcat ohne Security Manager und schauen, was passiert. Dazu benutzen Sie bitte unsere kleine Webanwendung *securitycheck.war* von der DVD in der mitgelieferten Tomcat-Umgebung *webdev-server*. Wir benutzen für jeden virtuellen Host ein getrenntes *${docBase}* im Verzeichnis *hosts/localhost/webapps*, wo sich auch die anzugreifende *ROOT*-Anwendung befindet.

Wenn Sie die Seite http://localhost:7380/securitycheck aufrufen, sehen Sie die folgenden Sicherheits-Checks (Abb. 1 und 2):

- In der Rubrik *Anzeige von fremden Dateien* stecken Versuche, die *ROOT/WEB-INF/web.xml*, *conf/tomcat-user.xml* und *conf/server.xml* aus dem Tomcat-Server anzuzeigen. Wie Sie sehen, klappt dies ohne Probleme, was eine Verletzung der ersten unserer Sicherheitsregeln darstellt.
- Mit der Rubrik „Neue Inhalte“ können Sie selber versuchen, in die *ROOT*-Anwendung eine neue JSP zu schreiben und zu editieren. Wie Sie sehen, ist auch dies kein Problem. Diese Aktion widerspricht allem, was uns heilig ist.
- Nun weiter: In der Rubrik *System-Eigenschaften aktiv angepasst* können Sie die System-Proxy-Einstellung ändern und somit die Web Services Ihrer Mitbewerber einem aktiven Scan unterziehen.
- Es bleibt Ihnen auch nichts erspart: Die Rubrik *Fremde Services* stellt eine Socket-Verbindung über den Port 110 her und könnte versuchen, die Mails anderer zu lesen. Auch das klappt problemlos, wenn dort ein entsprechender Mailserver läuft, und ist ebenfalls eine Verletzung unserer Sicherheitsbemühung.

Unsere Burg hat also derzeit alle Tore weit geöffnet und jeder Feind ist „noch“ auf das Herzlichste willkommen. Falls Sie sich jetzt schon auf eine Niederlage einstellen, weit gefehlt. Hilfe ist unterwegs und wird uns vor der absoluten und bedingungslosen Kapitulation erretten.

Um eine Besserung der aktuellen, augenscheinlich etwas unbefriedigenden Sicherheitssituation zu schaffen, stoppen wir den Tomcat-Server. Nutzen Sie zu

Abb. 2: Ohne Sicherheit lassen sich beliebige Dateien lesen und im Web veröffentlichen.

diesem Zweck ruhig einmal die „Böse Überraschung“ (siehe Listing unten). Alternativ lässt sich natürlich auch das klassische und etwas freundlichere Shutdown verwenden. Nach der oben aufgezeigten Niederlage in der virtuellen Schlacht um Ihre Tomcat-Burg wollen Sie Ihr „Glück“ sicherlich noch einmal mit aktiviertem Security Manager versuchen, oder?

In der Policy-Datei *conf/catalina.policy* sind unsere Sicherheitsregeln definiert, mit denen auch Struts-Anwendungen hoffentlich keine Probleme haben dürften. Um den Tomcat-Server mit aktivierter Sicherheit erneut zu starten, reicht es aus, den Parameter *-security* an den Startup-Befehl anzuhängen (Start des Tomcats mit aktiviertem Security Manager):

```
bin/startup.sh -security   (Linux/Unix)
bin/startup.bat -security  (Windows)
```

Was bewirkt der zusätzliche Parameter? In dem Tomcat-Startscript *$catalina.home/bin/catalina* werden dem Aufruf der JVM noch zwei Systemeigenschaften (*-D*-Optionen) hinzugefügt (zusätzliche Optionen für die JVM, um den Security Manager zu aktivieren):

```
-Djava.security.Manager
-Djava.security.policy=
                ${catalina.base}/conf/catalina.policy
```

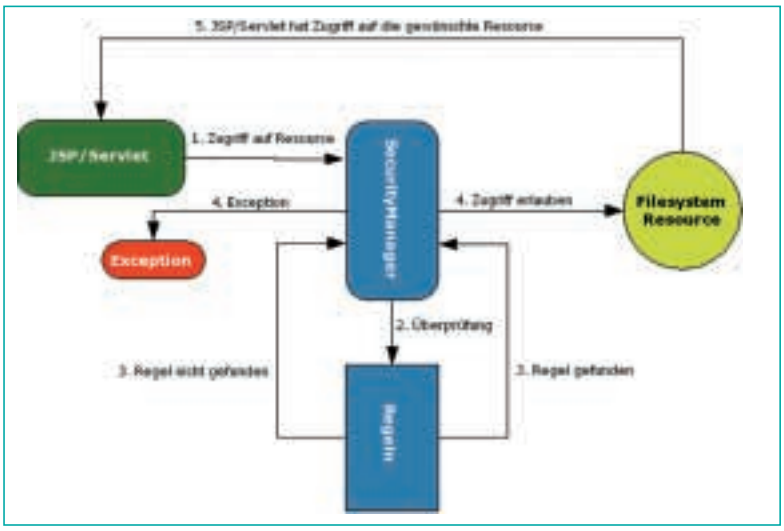
Solange die JVM jetzt läuft, werden nur Aktionen zugelassen, die in der angegebenen Policy-Datei – hier *catalina.policy* –

explizit erlaubt sind, wie Abbildung 3 verdeutlicht [2].

Wenn Sie jetzt die im ersten Versuch bereits durchgeführten Angriffsaktionen noch einmal ausprobieren, dann werden Sie bei jedem Verstoß gegen die angegebenen Sicherheitsregeln eine *SecurityException* bekommen. Anhand dieses Verhaltens können Sie erkennen, dass der Security Manager seine Arbeit wie gewünscht verrichtet.

Die Tore der Trutzburg sind geschlossen und der Feind sieht uns von den Burgzinnen grimmig grinsen. Die Hilfe in der Not kam – in Form des Security Managers und entsprechender Regeln – schnell und effektiv.

Abb. 3: Funktionsweise des Security Managers



Und was nun?

Wir haben unser Ziel erreicht. Die einzelnen Webanwendungen dürfen nur diejenigen Dinge tun, die Sie ihnen erlauben. Natürlich möchten wir nicht verschweigen, dass das Arbeiten mit Policies manchmal zum Haareraufen ist. Besonders erstaunlich – wenn nicht sogar erschreckend – ist es dabei, dass die Option des Startens mit dem Security Manager kein Standard beim Tomcat-Server ist. Aus Gründen der Fairness muss an dieser Stelle erwähnt werden, dass diese Aussage leider auch für die anderen J2EE-Server gilt.

Als ganz extrem negatives Beispiel und lästiges Ärgernis ist die folgende – oben bereits als „Böse Überraschung" erwähnte – JSP anzusehen:

```
<%
        System.exit(0);
%>
```

Ohne Security Manager würde die Ausführung dieser JSP den Tomcat sofort herunterfahren. Sie können sich vorstellen, wie oft so etwas passiert, wenn ein Provider mit einem seiner Kunden im Streit liegt. Schnell ist die Burg nicht erreichbar und gibt einer Servicemängelrüge weiteren Vorschub.

Tipps und Tricks

Wenn Sie Policies auf Windows- und Unix-Servern verwalten, kennen Sie sicherlich das Problem des netten Pfadspieles mit „\" und „/". Bei Pfadangaben für die *codebase* benutzen Sie immer „/", egal welches Betriebssystem zum Einsatz kommt. Diese Methode klappt bei den Pfadangaben für die *java.io.FilePermission* leider nicht immer. In einer solchen Situation hilft dann, um unabhängig vom Betriebsystem zu sein, der Einsatz der Property *${file.separator}* statt der normalen Dateitrennzeichen. Ein Wunder an Übersichtlichkeit wird Ihr Regelwerk dadurch allerdings bestimmt nicht werden. Deswegen gibt es für diese Property noch den Shortcut *${/}*.

Tückisch ist es auch herauszufinden, welchen legalen Zugriff das Regelwerk enthalten soll. Meist sieht man einer Weban-

wendung ihre Sicherheitsbedürfnisse nicht immer sofort an. Mit der Zeit sammelt man Erfahrung, welche Frameworks lesend auf Properties oder Ressourcen zugreifen. Häufig zeigt die gesamte Anwendung aber erst beim realen Einsatz ihre wirklichen Bedürfnisse. Zu einem Verstoß kommt es halt erst, wenn der entsprechende Code auch wirklich ausgeführt wird. Das Monitoring und Aufspüren solcher Situationen sind schwierig und nur mit entsprechender Geduld und manchmal mit den angeschalteten Sicherheits-Debug- ging-Optionen (*-Djava.security.debug=all*) zu finden [1].

Wenn Sie eine neue Anwendung installieren oder auch nur eine Änderung der Policy vornehmen, dann müssen Sie jedes Mal den Tomcat bzw. die JVM neu starten. Leider gibt es keine Konfiguration innerhalb des Tomcats, der die Policy „on the fly" neu lädt, obwohl das mithilfe des Security Managers kein wirk- liches Problem darstellen würde. Es scheint auch aus Sicht der Tomcat Commiter derzeit keinerlei Aktivitäten zu geben, etwas gegen diesen Missstand zu unternehmen.

Bedenken Sie auch, dass die Bibliotheken, die Sie in den Verzeichnissen *server/lib*, *common/lib* oder *shared/lib* hinzufügen, generell die Berechtigung *java.security.AllPermission* bekommen. Diese Bibliotheken dürfen innerhalb des Tomcats somit also alles. Kopieren Sie möglichst nur vertrauenswürdige Bibliotheken in dieses Verzeichnis. Hier wäre es sicherlich hilfreich, nur noch signierte Jar-Archive einzusetzen. Diese Aussage gewinnt umso mehr an Gewicht, wenn man bedenkt, dass Sie signierten Bibliotheken gezielt mehr Rechte erteilen können als ihren unsignierten Pendants (Listing 1).

Listing 1
Verschiedene Zugriffsrechte für signierte und unsignierte Bibliotheken

```
keystore "http://foo.bar.com/blah/.keystore";
//Der keystore-Eintrag steht immer am Anfang der Policy...
...
...
grant principal "alice" {
   permission java.io.FilePermission "/tmp/-", "read,
                                                write";
};

grant codebase "codebase"{
            permission java.io.FilePermission "/tmp/-",
                                                "read";
}
```

Solch ein Einsatz macht Sinn, da diese Bibliotheken dann signiert sind und Sie die Kontrolle über die von Ihnen gewünschte Sicherheit behalten. Drängen Sie bitte als Administrator und/oder Web- hoster Ihre Kunden darauf, schon mit eingeschalteter Policy zu entwickeln. Das erspart nach dem Deployment der Anwendung böse Überraschungen und schafft schon in der Entstehung eine stabile Betriebsumgebung mit entsprechendem vollständigen Sicherheitsregelwerk. Im Idealfall stellen Sie Ihren Kunden die Original-Policy des Servers zur Verfügung. Im Gegenzug dazu gibt der Kunde Ihnen sein Sicherheitsregelwerk bereits vor dem nächsten Deployment. Dieser Technik haben wir den Namen „fairer" Handel zwischen der Trutzburg und den Dörfern gegeben.

Ausblick

In dieser Kolumne haben Sie gesehen, wie Sie mit einfachen Schritten Ihren Tomcat vor unbefugten Zugriffen zu einer Trutzburg – von innen – absichern können. Dank des ausgereiften Sicherheitssystems der JVM mit dem Security Manager können viele Probleme sehr elegant eliminiert werden. Natürlich muss Ihr System zusätzlich mit Dateirechten und Firewall ausgestattet sein [6]. Was nutzt all die interne Sicherheit, wenn der Tomcat doch mit Root-Rechten auf dem System gestartet wurde? Wir hoffen, dass Ihnen diese Kolumne auch in der Argumentation oder dem Streit um den Einsatz von Java-Servern dienen kann. Die Klassiker von cgi, Perl, PHP bis ASP machen, nach unserer Auffassung, auf dem Gebiet der inneren Sicherheit leider überhaupt keine gute Figur.

Die Grundregel der Sicherheit lautet: erst einmal sperren, dann prüfen und gegebenenfalls freischalten! Machen Sie sich ruhig etwas mehr Arbeit zu Beginn mit genaueren Sicherheitsregeln, das spart Ärger mit den Entwicklern und verhindert lästige Betriebsunterbrechungen in der Zukunft. Mit einem gut strukturierten und geprüften Regelwerk haben Sie dann im Angriffsfall weniger Probleme und verlieren nicht den Spaß in Ihrer Burg.

Weitere Sicherheitsmaßnahmen des Tomcats wie SSL, Realms, spezielle Anfragefilter, Logger und AccessLogger können Ihre Webanwendungen letztendlich wirklich absichern und lassen die Seite für alle kontrollierbar werden [7], [4].

Im Übrigen sollten Sie unsere „Trutzburg" natürlich ausschließlich als eine Demo betrachten und weitere feindliche Gedanken für alle Zeiten gleich wieder vergessen. Eigentlich haben Sie diesen Beitrag und das Beispiel nie zur Kenntnis genommen. Im Prinzip unterliegt alles Beschriebene der absoluten Geheimhaltung der Gemeinschaft der Tomcat-Burgbesitzer. Gegen feindliche Übernahmen anderer, nicht geschützter Burgen anderer Projekte oder Hersteller darf dieses Ideengut natürlich – leider – nicht verwendet werden.

Wir freuen und schon auf Kommentare und Anregungen – besuchen Sie also unsere TomC@-Site [8] und das TomC@-Forum [9] oder die Tomcat-Mailing-Listen [10].

Peter Roßbach (pr@objektpark.de) ist als freier J2EE-Systemarchitekt, Entwickler und Trainer tätig. Lars Röwekamp ist als IT-Berater mit den Schwerpunkten Neue Technologien und OO/Enterprise Java für seine Firma OpenKnowledge tätig. Thorsten Kamann (thorsten.kamann@planets.de) ist als Softwareentwickler und Administrator für Win32- und Linux-Systeme tätig

Links & Literatur

[1] Bruce Sams: Java Security, Teil 1: Security Maßnahmen aktivieren und konfigurieren in *Java-Magazin* 11.2003

[2] java.sun.com/j2se/1.4.2/docs/guide/security/PolicyFiles.html

[3] 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

[4] Vivek Chopra, Ben Galbraith, Gothamm Polisetty, Brain P. Rickabaugh, John Turner: Apache Tomcat Security Handbook, Wrox Press, 2003, S. 51 ff.

[5] java.sun.com/j2se/1.4.2/docs/guide/security/permissions.html

[6] Sicheres Linux. Schutz für Ihre Daten, Konzepte, Methoden, Werkzeuge, in Linux New Media AG, Sonderheft Linux Magazin 1/2004

[7] java.sun.com/webservices/docs/1.0/tutorial/doc/WebAppSecurity.html

[8] tomcat.objektpark.org/

[9] www.javamagazin.de/tomcat/

[10] jakarta.apache.org/tomcat/