Benjamin Emde (734787)

Apache Tomcat

„a chain is only as strong as its weakest link“

Inhaltsverzeichnis

Die Sicherheit von Webanwendungen ist in Zeiten von ebay, amazon und studiVZ zu einem entscheidenden Thema geworden. Das Seminar Secure Web Application Engineering befasst sich mit eben dieser Problematik. Eine sichere Anwendung, verwirklicht durch ein verstärktes Bewusstsein für die möglichen Schwachstellen schon während der Entwicklung, soll das Ziel sein. In diesem Kontext werden verschiedene Technologien betrachtet. Diese Ausarbeitung handelt von den Sicherheitsaspekten des Servlet-Containers Apache Tomcat.

Einleitung

Der Apache Tomcat ist die offizielle Referenzimplementierung der Servlet-Spezifikation. Ein Servlet ist eine Java-Klasse die zur Erweiterung der Serverfunktionalität dient. Tomcat bietet damit einen Rahmen um Servlets einzusetzen. Er kann analog zum Apache HTTPD als Webserver eingesetzt werden. Üblich ist jedoch der Einsatz als Add-On zu einem gewöhnlichen Webserver. Die Vorteile dessen werden im nächsten Abschnitt erläutert. Außerdem kann der Tomcat auch eingebettet in andere Java-Anwendungen eingesetzt werden. Darüber hinaus ist in Tomcat ein JavaServer Pages (JSP) Compiler integriert.
Die in diesem Dokument beschriebene Evaluierung des Apache Tomcat bezieht sich auf die Version 6.0, welche die Spezifikationen 2.5 für Servlets und 2.1 für JSPs implementiert ([Apache]).
Servlets sind ein offener Standard definiert von Sun Microsystems. Sie leiten sich normalerweise von der generischen oder HTTP-spezifischen Implementierung des Servlet-Interface ab. Zur Ausführung ist eine Java Virtual Machine (JVM) notwendig, die durch den Container bereitgestellt wird. Es wird keine JVM im Browser des Clients benötigt. Eine vergleichbare Technologie beschreibt das Common Gateway Interface (CGI), welches ebenfalls ermöglicht unabhängig vom eingesetzten Browser eine Webseite dynamisch zu generieren, indem externe Software eingebunden wird. Ein klarer Vorteil von Servlets gegenüber CGI ist die Effizienz. Bei jeder Anfrage wird statt einem neuen Prozess nur ein leichtgewichtiger Java Thread erzeugt. Darüber hinaus werden Servlets unabhängig von den Spezifika des Webservers geschrieben und sind damit portabel wie jede andere Java Klasse. Außerdem sind die Größe der Java Community und die damit verbundenen Vorteile nicht zu unterschätzen. Bevor auf die sicherheitsfördernden Aspekte eingegangen wird, wird noch kurz dargestellt wie Servlets heute zum Einsatz kommen.

```
import javax.servlet.*;
public class MyServlet extends GenericServlet {
        public void service (
                ServletRequest request,
                ServletResponse  response
        ) throws ServletException, IOException
        {
                ...
        }
        ...
}
```

Abbildung 1: Aufbau eines Servlet

Der ebenfalls von Sun definierte Standard JavaServer Pages ermöglicht die Trennung der Präsentationsschicht von der Logik einer Webanwendung und damit die Unabhängigkeit des Designers vom Entwickler. JSPs werden vom Container zu Servlets compiliert und sind einfach gesagt HTML mit eingebettetem Java Code.
Mittlerweile gibt es zahlreiche auf Servlets aufbauende Technologien, die dem Entwickler noch einfacher und schneller ermöglichen eine dynamische Webanwendung zu erstellen. Bekanntere Frameworks sind Struts, JavaServer Faces, Spring MVC sowie Grails – eine interessante Verknüpfung der Einfachheit von Ruby on Rails mit der Sicherheit von Servlets.

Sicherheit durch JVM und Java EE

Java bietet von Haus aus mehrere Technologien zur Verbesserung der Sicherheit einer Anwendung an. Neben automatischer Speicherverwaltung, statischer Typisierung und eingebautem Exception Handling stehen viele Standard APIs zur Verfügung. Der Apache Tomcat versucht nicht nur der Servlet-Spezifikation zu genügen, sondern ermöglicht auch den Einsatz vieler Teile der Java Platform, Enterprise Edition (Java

EE). Das betrifft APIs wie JNDI und JAAS, welche später noch vorgestellt werden. Der eingebaute Java Security Manager besitzt erweiterte Funktionalität. Er filtert den Zugriff der Anwendung auf Systemressourcen. Entsprechende Regeln sind in einer Policy-Datei zu definieren.

Sicherheit durch Tomcat

Die Sicherheitsstrategien des Apache Tomcat können unterteilt werden in einen deklarativen und einen programmatischen Teil, wobei letzterer optional ist, also eine „second line of defense“ darstellt.
Die Sicherheitspolitik wird vor allem in den folgenden 4 Dateien deklariert: Die server.xml, web.xml, catalina.policy und catalina.properties. Zu finden sind diese im Verzeichnis `<$CATALINA_HOME>/conf/`. Hier werden Standardeinstellungen vorgenommen, die zunächst für alle Webanwendungen gelten. In der server.xml werden allgemeine Einstellungen der Engine und der verwendeten TCP-Ports, Konfigurationen von Connectoren, Protokollierung, Load-Balancing etc. vorgenommen. In der web.xml werden Einstellungen für Webanwendungen wie z.B. die Zuordnung von URLs zu Servlets bzw. JSPs definiert. Die Datei catalina.policy bestimmt dateispezifisch die oben angesprochenen Regeln für den Java Security Manager. In Verbindung mit catalina.properties können Zugriffsrechte für ganze Java-Pakete deklariert werden. Zu beachten ist, dass der sogenannte “Deployment Descriptor” (web.xml) zusätzlich auf weiteren Ebenen platziert werden kann, wobei die Regeln der tieferen Ebene die der höheren überschreiben. Die Ebenen müssen in der server.xml definiert werden und spiegeln sich in der Verzeichnisstruktur des Tomcat wider. Abbildung 2 zeigt einen möglichen Aufbau der Ebenen des Apache Tomcat.

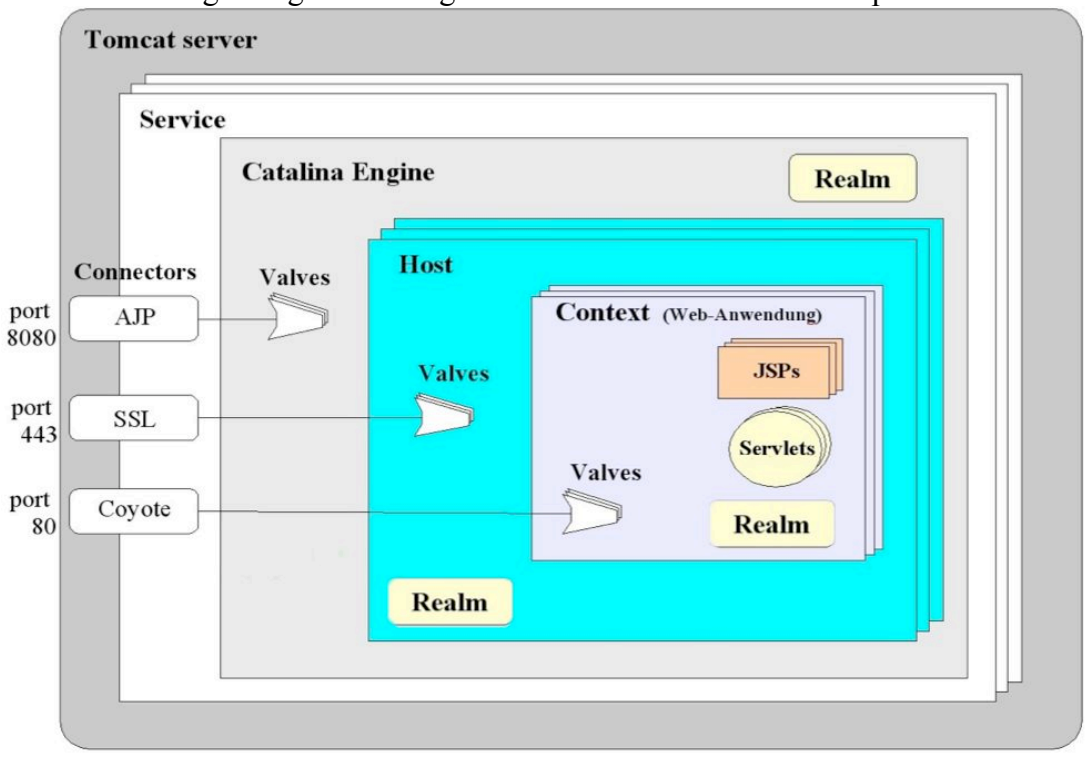

Abbildung 2: Architektur des Apache Tomcat, leicht verändert aus [Chopra]

Ein Server beinhaltet mehrere Services, diese umfassen jeweils eine Sammlung von Connectoren und eine Engine. Durch die Connectoren kommuniziert entweder ein Browser, im Falle des Coyote direkt, mittels Secure Sockets Layer (SSL) über eine gesicherte Verbindung, oder es kommuniziert ein Webserver über das Apache JServ

Protocol (AJP) mit dem Tomcat. Eine Engine wiederrum kann mehrere Hosts, die mit einer Website verbunden sein können, enthalten. Ein Host beherbergt beliebig viele Contexts, welche die eigentliche Webanwendung bestehend aus Servlets und JSPs repräsentieren.
Eine Besonderheit sind die Komponenten Valve und Realm. Sie können in den 3 gezeigten Ebenen Engine, Host und Context platziert werden und haben dadurch

```
<Host name="localhost" ...>
   ...
   <Valve className="org.apache.catalina.valves.RemoteAddrValve“
          deny="192.168.1.*"/>
   ...
</Host>
```

Abbildung 3: Deklarierung eines Request Filter Valve in der server.xml

verschiedene Reichweiten. Somit gibt es mehrere Möglichkeiten den eingehenden Datenverkehr zu filtern (siehe Abbildung 3), protokollieren (siehe Abbildung 4) oder ähnliches. Ein Realm definiert einen Bereich auf den bestimmte Benutzer Zugriff haben. Er bestimmt insbesondere wie die Authentisierungsdaten der Benutzer

```
<Engine name="Standalone" ...>
   ...
   <Valve className="org.apache.catalina.valves.AccessLogValve“
          prefix="catalina_access_log." suffix=".txt“
          pattern="common"/>
   ...
</Engine>
```

Abbildung 4: Deklarierung eines Access Log Valve in der server.xml

gespeichert werden.
Mittels des Deployment Descriptors kann der Zugriff von außen rollenbasiert beschränkt werden. Hierzu werden die Tags `<security-constraint>` in Verbindung mit `<security-role>` verwendet. Ersteres bestimmt eine Abbildung von Rollen auf URL-Pattern für die der Zugriff, ggf. durch Erfüllung der im Tag `<user-data-constraint>` deklarierten Vorraussetzungen, gewährt wird. Letzteres definiert die Rollen an sich.
Wesentlich feiner kann eine Webanwendung nur programmatisch gesichert werden. In den Servlets werden üblicherweise die Methoden `getRemoteUser`, `isUserInRole` und `getUserPrincipal` verwendet um einzelne Abschnitte zu sichern.

Sicherheitslücken der Standardkonfiguration

Die oben erwähnten Standardeinstellungen, die innerhalb der Dateien des Verzeichnisses `<$CATALINA_HOME>/conf/` vorgenommen wurden, sollen im Folgenden beleuchtet werden. Das seit Version 5.0.28 des Tomcat im Default Deployment Descriptor auskommentierte `InvokerServlet` ermöglicht den direkten Aufruf von Servlets, die noch nicht in einer web.xml deklariert, aber bereits im Tomcat Verzeichnis vorhanden sind ([BSI]). Dieses auf den ersten Blick vereinfachende Verfahren umgeht sämtliche Sicherheitsbeschränkungen und ist damit potentiell gefährlich.
Die in der Policy-Datei für den Java Security Manager deklarierten Einschränkungen des Zugriffs von Code auf Systemressourcen sind nur aktiv, wenn der Tomcat Server mit

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

Abbildung 5: Zuweisung einer Permission in der catalina.policy

`startup -security` gestartet wird.
Davon abgesehen wurden viele sinnvolle Einstellungen vorgegeben. Für die Verwendung von Server Side Includes (SSI) sowie für den Einsatz von Legacy-Komponenten mittels CGI wurden bereits Standards deklariert. Diese müssen nur noch einkommentiert werden.

Identitätsmanagement am Beispiel Tomcat

Die Zugriffskontrolle des Apache Tomcat ist wie oben erwähnt rollenbasiert. Es werden die folgenden 3 Schritte durchlaufen. Zunächst erfolgt die Übertragung der Authentisierungsdaten, dann die Überprüfung der Daten und letztlich die Zuweisung einer Rolle bzw. die Ausgabe einer Fehlermeldung.
Anfangs muss also Benutzername und Passwort übertragen werden, dazu kann eine von 4 Möglichkeiten gewählt werden, die mit dem Tag `<auth-method>` in der web.xml festgelegt wird. Base64-kodiert werden die Daten bei der HTTP-BASIC Methode gesendet, als sicherer MD5-Hash wenn die HTTP-DIGEST Methode verwendet wird und zusammen mit einem Client-Zertifikat über eine gesicherte Verbindung wenn SSL anhand des Symbols `CLIENT-CERT` aktiviert wurde. Viertens kann die Formbasierte Authentisierung benutzt werden, welche standardmäßig ebenfalls unverschlüsselt sendet jedoch ohne weiteres Hash-Methoden und SSL verwenden kann. Nur hierbei kann der Entwickler die Login-Seite selbst gestalten. Ein Logout ist auch nur hierbei ohne Beendigung des Browsers möglich.
Die Überprüfung der Benutzerdaten geschieht i.d.R. indem Tomcat diese mit den im Security-Realm gespeicherten Daten vergleicht. Für die Speicherung sind im Wesentlichen die folgenden 4 Varianten üblich.
Die einfachste Möglichkeit, Benutzername, Passwort und zugehörige Rolle zu archivieren, ist die Nutzung der tomcat-users.xml. In dieser Datei können mittels XML-Tags die Daten, standardmäßig auch das Passwort, im Klartext abgespeichert werden. Allerdings können mit der Einstellung `digest` in der server.xml auch entsprechende Hash-Werte gespeichert werden.
Genauso gut kann auch eine relationale Datenbank zur Verwaltung der Daten verwendet werden. Die Datenbank wird dazu mit Hilfe der Java Database Connectivity (JDBC) API kontaktiert. Hierzu benötigt diese einen JDBC-Treiber.
Die dritte Option ist die Verwendung eines Verzeichnisdiensts der über die Java Naming and Directory Interface (JNDI) API angesprochen wird. In diesem Fall ist auch eine unter Umständen sicherere Authentisierung durch den Verzeichnisdienst selbst denkbar (sog. `bind` Modus). Im `comparison` Modus hingegen holt sich der Tomcat wie gehabt die Daten zum Vergleich.
Die letzte Alternative ist die Verwendung der Java Authentication and Authorization Service (JAAS) API. Diese schafft eine Abstraktionsebene zwischen der Webanwendung und der Technologie zur Authentisierung. JAAS gibt nur ein Rahmenwerk vor, die eigentliche Authentisierungslogik ist in LoginModules ausgelagert und kann unabhängig vom Code der Webanwedung ausgetauscht werden. Diese Herangehensweise ähnelt der bei Betriebssystemen angesiedelten Technologie der Pluggable Authentication Module (PAM). Direkt in JAAS enthalten sind Module für die Authentisierung mit JNDI, Kerberos, den Windows NT und Solaris Benutzerdaten.
Single Sign-On ermöglicht der Apache Tomcat von Haus aus nur Host-basiert, d.h. dass ein Benutzer sich an einem Context authentisiert und im Laufe einer Session auf alle weiteren Contexts unter dem selben Host Zugriff hat. Dazu muss der Security-Realm auf der Ebene der Engine oder des Hosts definiert werden, damit ein Nutzer

```
<Host name="localhost" ...>
   ...
   <Valve className="org.apache.catalina.authenticator.SingleSignOn“
          debug="0"/>
   ...
</Host>
```

Abbildung 6: Deklarierung von Single Sign-On in der server.xml

immer mit dem selben Namen und Passwort gespeichert ist. Die Authentisierungsdaten werden daraufhin für alle Webanwendungen des Hosts gecacht.
Neue mit Identity 2.0 entstandene Technologien sind, aufgrund der großen Community, mit einer großen Wahrscheinlichkeit auch schon in Java implementiert. Im Falle von OpenID ist das, durch SXIP initiiert und mittlerweile von Google unterstützt, durch das Projekt OpenID4Java geschehen (www.openid4java.org).

Fazit

Der Servlet-Container Apache Tomcat ermöglicht zahlreiche Sicherheitsfunktionen. Durch den Einsatz von Java ergeben sich einige Vorteile gegenüber konventionellen Scriptsprachen. Nicht zuletzt fördert die große Community das Suchen und Beseitigen von Sicherheitslücken. Zudem erreicht der Tomcat die Trennung zwischen Sicherheitspolitik und Logik der Webanwendung. Zu bemängeln ist jedoch die fehlende Integration von Single Sign-On mit anderen Webservern. Das Deployment einer Webanwendung bedarf der mühsamen Konfiguration von server.xml und den web.xml. Alles in allem verlangt der Tomcat bzw. Java jedoch viel Code für wenig Funktionalität, verglichen mit Ruby on Rails. Es kann leicht die Übersicht verloren gehen, wenn nicht entsprechende Tools zum Einsatz kommen. Von Apache vorgegeben sind die Webanwendungen admin und manager, mit der zur Laufzeit deployed und undeployed werden kann. Noch mehr Informationen liefert das Monitoring Tool Lambda Probe (www.lambdaprobe.org). Weniger „Lines of Code“ bieten weniger Möglichkeiten Fehler einzubauen. Dafür sinkt unter Umständen die Wartbarkeit und Skalierbarkeit der Anwendung. Somit bleibt die Frage ab welcher Projektgröße der Tomcat eingesetzt werden sollte. Abschließend lässt sich sagen vor allem für sicherheitskritische Webanwendungen ist Tomcat eine gute Wahl, wenn auf proprietäre Software verzichtet werden soll.

Quellen

[Apache] *Apache Tomcat.* http://tomcat.apache.org/ . Stand: 02.12.2007

[JSP Sun] *JavaServer Pages[tm] Technology - White Paper.* http://java.sun.com/products/jsp/whitepaper.html . Stand: 02.12.2007

[Hunter] Jason Hunter, William Crawford. *Java Servlet Programming*. O'Reilly. Sebastopol, CA. 2001

[Geary] David M. Geary. *Advanced JavaServer Pages*. Sun Microsystems Press. Upper Saddle River, NJ. 2001

[Cams] *Tomcat Security Overview and Analysis.* http://www.cafesoft.com/products/cams/tomcat-security.html . Stand: 02.12.2007

[BSI] *Sicherheitsuntersuchung des Apache Jakarta Tomcat Servlet Containers.* Bundesamt für Sicherheit in der Informationstechnik. Bonn. 2006

[Chopra] V. Chopra et al. *Professional Apache Tomcat 5*. J. Wiley Publishing. Indianapolis, IN. 2004