Java Torwächter – universelle Sicherheit mit Acegi Security

Autor: Ömer Gürsoy, Opitz Consulting Gummersbach GmbH

In vielen Software-Projekten spielt Sicherheit eine wichtige Rolle. Um in Geschäftsanwendungen eine wartbare Architektur beizubehalten, werden Authentifizierung und Autorisierung als nicht funktionale, technische Anforderungen von der eigentlichen Geschäftslogik separiert. Mit dem Acegi-Security-Framework steht dazu ein mächtiges und flexibles Werkzeug zur Verfügung, das sowohl deklarativ als auch programmatisch für die Sicherung von Java-basierten Web-, Desktop- und Server-Anwendungen sorgt. Dieser Artikel gibt einen Überblick über die allgemeine Funktionsweise von Acegi Security, seine Einsatzmöglichkeiten und die Unterschiede zu alternativen Lösungen.

Das Acegi-Security-Framework ist 2003 durch eine Anfrage an die Spring-Entwickler nach einer Spring-basierten Sicherheitslösung entstanden. Eine kleine Gruppe entwickelte den daraufhin erstellten Entwurf weiter und veröffentlichte die Quellen im März 2004 als selbstständiges SourceForge-Projekt unter der Apache-Lizenz. Weiter ausgereift bekam das Projekt den Status eines offiziellen Spring-Framework-Unterprojekts. Im Mai 2006 erschien das erste stabile Release.

Wesentliche Merkmale

Das Acegi-Security-Framework zeichnet sich durch folgende Merkmale aus:

- eine robuste und ausgereifte Architektur für den Umgang mit Authentifizierung und Autorisierung
- eine hochgradige Konfigurier- und Individualisierbarkeit durch die aus dem Vorbild-Projekt Spring geerbten Designprinzipien DI (Dependency Injection) und AOP (Aspect Oriented Programming)
- eine Fülle an unterstützten Verfahren und Kopplungsmöglichkeiten an wichtige Systeme (zum Beispiel SSO-Systeme, Servlet-Container)
- eine sehr gute Dokumentation
- eine hohe Akzeptanz (große Community, Einsatz in Banken etc.)

Allgemeine Funktionsweise

Das Sichern von Ressourcen funktioniert deklarativ durch Beobachten von Methoden-Aufrufen oder HTTP-Anfragen. Imperativ (programmatisch) kann hingegen an beliebiger Stelle mit den Framework-Komponenten gearbeitet werden. Besonders interessant sind die deklarativen Möglichkeiten, die es erlauben, den Code der Sicherung zu zentralisieren und den der Geschäftslogik unangetastet zu lassen. Zur Sicherung von Methoden-Aufrufen kommt hier die Aspekt-orientierte Programmierung (AOP) zum Einsatz. Deren Vorgehensweise ist vergleichbar mit jener von Datenbank-Triggern: Ausgewählte Aktionen lassen sich in einem so genannten Advice zusammenfassen und deklarativ bestimmen. Dieser Advice wird vor und/oder nach dem

Aufruf einer Methode durchgeführt. So lassen sich beispielsweise die Authentifizierung und Autorisierung vor dem Aufruf prüfen beziehungsweise der Rückgabewert nach dem Aufruf verändern. Genauso können HTTP-Anfragen im Vor- und/oder Nachhinein geprüft beziehungsweise manipuliert werden. Dabei kommen einfache Web-Filter zum Einsatz.

Unter anderen werden folgende Technologien zur Authentifizierung unterstützt:

- HTTP BASIC authentication headers
- HTTP Digest authentication headers
- HTTP X.509 client certificate exchange
- LDAP
- Form-based authentication
- CA Siteminder
- JA-SIG Central Authentication Service (CAS)
- Java Authentication and Authorization Service (JAAS)

Eine individuelle Lösung lässt sich mit geringem Aufwand über eine eigene Implementierung einsetzen.

Das Orchestrieren der letztlich eingesetzten Implementierungen erfolgt mittels Dependency-Injection. Das bedeutet, dass die Abhängigkeiten von Objekten so konfiguriert sind, dass sie erst zur Laufzeit in das abhängige Objekt injiziert werden. Die Flexibilität spiegelt sich auch in der Grund-Architektur wider, die abstrakt und erweiterbar gehalten ist.

Wichtige Komponenten und ihre Funktionen

Für die Steuerung des Frameworks ist der AbstractSecurityInterceptor zuständig, der in drei Ausprägungen existiert: Der FilterSecurityInterceptor dient der Behandlung von HTTP-Anfragen. Der AspectJSecurityInterceptor und der MethodSecurityInterceptor sind AOP-basierte Implementierungen zur Behandlung von Methoden-Aufrufen, die jeweils verschiedene AOP-Implementierungen (AspectJ beziehungsweise AOP Alliance) nutzen.

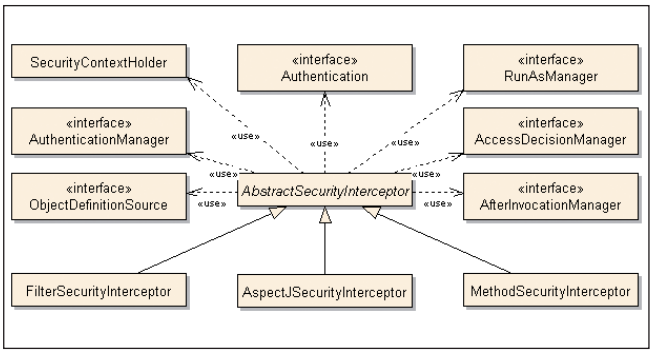

Abbildung 1: Klassendiagramm mit einem Ausschnitt der wichtigsten Komponenten

Die ObjectDefinitionSource enthält Angaben darüber, welche Ressource mit welcher Rolle geschützt ist.

Zur Authentifizierung wird ein Authentication-Objekt erzeugt, das von dem SecurityContextHolder gehalten wird. Dieser unterstützt drei Strategien: In der Standardeinstellung legt der SecurityContextHolder das Authentication-Objekt im lokalen Thread ab. Er kann dieses aber auch an Threads weitergeben, die aus dem gesicherten Thread erzeugt wurden oder es global in der JVM setzen.

Existiert in der aktuellen Benutzersitzung noch kein Authentication-Objekt, wird es unter Verwendung eines der konfigurierten Authentifizierungsverfahren erzeugt. Es ist auch möglich, mehrere Verfahren in einer Kette hintereinander zu schalten. So lässt sich zum Beispiel eine Anmeldung mit Zertifikat vor eine solche mit Benutzername und Passwort schalten. Konnte der Benutzer anhand eines an die Anfrage anhängenden Zertifikats authentifiziert werden, ist es die Anzeige des Login-Dialogs nicht mehr nötig. Sobald eine geschützte Ressource angesprochen wird, erfolgt die Prüfung, ob das Authentication-Objekt bereits authentifiziert ist und die nötige Rolle enthält. Ist das nicht der Fall, ist der AuthenticationManager dafür zuständig, das Authentication-Objekt gegen ein beliebiges Benutzer-Repository (LDAP, Datenbank, SSO-Systeme etc.) zu validieren und die Rollen zu ermitteln.

Mit der RunAsManager-Komponente können dem aktuellen Benutzer temporär weitere Rollen zugewiesen werden.

Der AccessDecisionManager ist für die Autorisierungsprüfung zuständig. Mit dem AfterInvocationManager können Rückgabewerte beobachtet werden, wobei sich bei unzureichender Berechtigung die Objekte verändern bzw. eine AccessDeniedException werfen lassen.

Vergleich zu alternativen Frameworks

Ein wesentlicher Unterschied zu JAAS besteht darin, dass das Acegi-Security-Framework mit der Anwendung ausgeliefert werden kann, ohne im Container integriert sein zu müssen. Hinzu kommt, dass es in der Grundausstattung mehr Verfahren unterstützt und sich einfacher und schneller an individuelle Anforderungen anpassen lässt. Weitere kostenfreie Alternativen sind JGuard (JAAS-basiert) und Kasai, die auf den ersten Blick vielversprechend erscheinen. Die fehlende praktische Erfahrung mit diesen Frameworks lässt jedoch an dieser Stelle keinen direkten Vergleich zu.

Fazit

Die Verwendung von Acegi Security ermöglicht die standardisierte Entwicklung und Wartung von Sicherheitsmechanismen in und über Unternehmensanwendungen hinweg. Bereits vor Erscheinen der ersten stabilen Version kam das Framework in zahlreichen Projekten erfolgreich zum Einsatz. Grundsätzlich ist festzuhalten, dass Java-Anwendungen in sehr kurzer Zeit mit Authentifizierung und Autorisierung versehen und in bestehende Sicherheitsmechanismen integriert werden können. An dieser Stelle muss wieder einmal auf die Verdienste der Open-Source-Community hingewiesen werden, ohne die ein solches Angebot an qualitativ hochwertiger, aktueller und kostenloser Software nicht existieren würde.

Kontakt:
Ömer Gürsoy
oemer.guersoy@opitz-consulting.de