## 9.1 Minimale Autokonfiguration

Während der `spring-boot-starter-security` in Spring-Boot-1-Anwendungen noch etliche Annahmen getroffen hat und viele Konfigurationseigenschaften bereitstellte, diese Annahmen zu überschreiben, so wurde dieses Verhalten mit Spring Boot 2 grundsätzlich geändert. Falls Sie Spring Boot Security 2 über den Starter einbinden, werden alle Web-Endpunkte – inklusive der Actuator-Endpunkte – geschützt. Der Starter stellt dabei einen Default-Benutzer bereit, dem – je nach angefragtem Inhaltstyp – Basic- oder Form-Login angeboten wird. Dieses Verhalten entspricht dem Standardverhalten von Spring Security. Damit stellt dieser Starter, der im Folgenden auch als Spring Boot Security 2 bezeichnet wird, nur eine minimale Autokonfiguration zur Verfügung.

Jegliche Konfiguration, die über die Abschaltung der vollständigen HTTP-Sicherheit mittels `security.basic.enabled = false` (für den Fall, dass Sie zum Beispiel nur Methodensicherheit nutzen möchten) und der Bereitstellung eigener Zugangsdaten hinausgeht, erfordert das Vorhandensein einer Bean vom Typ `WebSecurityConfigurerAdapter`. Sobald eine solche Bean im Kontext vorhanden ist, bewirkt der Starter lediglich ein grundsätzliches `@EnableWebSecurity` und stellt darüber hinausgehende Autokonfiguration ein. Diese Entscheidung wurde getroffen, damit Sie sich als Entwickler explizit mit der Konfiguration von Security beschäftigen müssen. Die Gefahr, etwas falsch zu konfigurieren oder von übrig gebliebenen Fragmenten der automatischen Konfiguration überrascht zu werden, ist im Bereich Security mit größerem Risiko verbunden.

`WebSecurityConfigurerAdapter` ist eine abstrakte Klasse und stellt Schnittstellen zur Verfügung, die als Parameter einen Builder zur Konfiguration von Spring Security erhalten.

## 9.2 Die Grundlagen verstehen

Spring Security nutzt folgende Hauptbestandteile, um die Themen Authentifizierung und Autorisierung zu realisieren:

- Der `SecurityContextHolder` realisiert den Zugriff auf den `SecurityContext`.
- Der `SecurityContext` beinhaltet die `Authentication`-Instanz sowie gegebenenfalls Request-spezifische Informationen.
- `Authentication` repräsentiert einen erfolgreich authentifizierten Principal.
- Ein oder mehrere `GrantedAuthority`-Instanzen beschreiben die Rechte einer `Authentication`.

- Eine Instanz des `UserDetailsService` stellt eine Schnittstelle bereit, um mittels eines Benutzernamens oder eines Zertifikats die `UserDetails` zu ermitteln, deren Authentizität überprüft wird.

Der `SecurityContextHolder` ist zentrales Element von Spring Security. In der Standardkonfiguration speichert diese Klasse alle Informationen, insbesondere über den aktuellen Principal in einem `ThreadLocal`: Damit stehen diese Informationen innerhalb des ausführenden Threads immer zur Verfügung. Spring Security trägt weiterhin Sorge dafür, dass der `ThreadLocal` nach Ablauf eines Requests bereinigt wird.

Damit ist auch klar, dass Spring Security nicht für alle Arten von Anwendungen ohne zusätzliche Konfiguration gleichermaßen gut geeignet ist: Anwendungen, in denen jeder Thread demselben Principal zugeordnet wird, zum Beispiel in einer JavaFX-Anwendung, oder reaktive Anwendungen, in denen Principals einer Kette von Ereignissen und nicht einzelnen Threads zugeordnet werden müssen, müssen gesondert betrachtet werden. In diesem Kapitel werden wir nur über Spring Security im Webkontext sprechen, allerdings sind die im Folgenden vorgestellten Konzepte unabhängig vom `SecurityContextHolder` gültig.

### 9.2.1 Authentifizierung

Spring Security ermöglicht Authentifizierung über folgende Technologien, die entweder Teil des Spring-Security-Projekts sind oder von Dritten bereitgestellt werden:

- HTTP-Basic-Authentifizierung
- HTTP-Digest-Authentifizierung
- HTTP X.509 client certificate exchange
- LDAP
- Form-based-Authentifizierung
- OpenID-Authentifizierung
- Java Authentication and Authorization Service (JAAS)
- und viele weitere mehr

Abbildung 9–1 zeigt die Kernkomponenten von Spring Security. Die Authentifizierung wird an Provider delegiert, die vollständig initialisierte `Authentication`-Objekte zurückgeben oder eine Exception werfen, wenn die Authentifizierung nicht erfolgreich war.

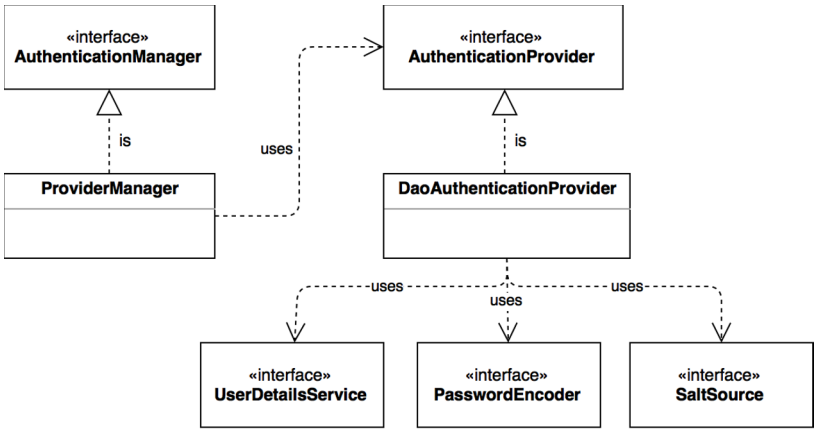


***Abb. 9–1***
*Kernkomponenten von Spring Security*

Spring Security stellt eine ganze Reihe von Implementierungen des `AuthenticationProvider` bereit, sowohl intern als auch über externe Module. So gibt es LDAP-, OpenID- oder Keycloak-spezifische Implementierungen. Der im Bild gezeigte `DaoAuthenticationProvider` ist eine der ältesten Implementierungen. Er nutzt einen `UserDetailsService`, um Benutzerdaten aus verschiedenen Quellen zu laden. Das Projekt selber stellt eine In-Memory- sowie eine Jdbc-Implementierung zur Verfügung.

Der `spring-boot-starter-security` konfiguriert für Sie einen `AuthenticationManager` mit einem In-Memory-`UserDetailsService`. Dieser Service enthält genau einen Benutzer. Ebenfalls konfiguriert werden Ereignisse, die über den in Abschnitt A.1 beschriebenen Mechanismus veröffentlicht werden. Zu den veröffentlichten Ereignissen gehören unter anderem `AuthenticationSuccessEvent` und `AbstractAuthenticationFailureEvent`, Letzteres in verschiedenen Ausprägungen. Zugriffe auf Ressourcen durch einen authentifizierten Principal ohne entsprechende Berechtigung lösen Access-Denied-Ereignisse aus.



### 9.2.2 Autorisierung

Autorisierung beziehungsweise die Durchsetzung von Berechtigungen wird üblicherweise als querschnittlicher Aspekt einer Anwendung betrachtet; innerhalb von fachlichem Code sollten explizite Abfragen, ob eine bestimmte Aktion erlaubt ist oder nicht, vermieden werden. Spring Security realisiert das in Hinblick auf Methodenaufrufe über Springs AOP-Support (siehe Abschnitt 3.2) und für HTTP-Requests über Standard-Servlet-Filter.

Abbildung 9–2 zeigt die Komponenten von Spring Security, die an der Entscheidungsfindung, ob ein Principal für ein bestimmtes Objekt autorisiert ist, beteiligt sind. Es wird eine von mehreren Implementie-

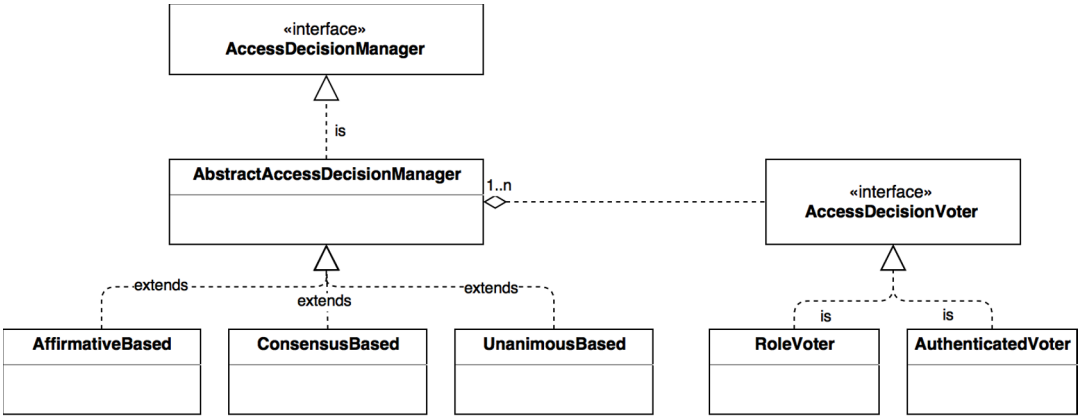


***Abb. 9–2***
*Entscheidungsfindung*

rungen des `AccessDecisionManager` genutzt, um zu entscheiden, ob ein Principal, repräsentiert durch ein `Authentication`-Objekt, Zugriff auf ein gesichertes Objekt hat oder nicht. Dabei kann die Entscheidung einheitlich, bestätigend oder über einen Konsens erfolgen. Wie eine Entscheidung getroffen wird, hängt dabei von einem oder mehreren `AccessDecisionVoter` ab. Diese Objekte können zum Beispiel auf den Authentifizierungsstatus oder die Rollen eines Principals zugreifen.

`spring-boot-starter-security` nutzt dieselbe Default-Konfiguration wie Spring Security und stellt für Ihre Anwendung einen `AffirmativeBased AccessDecisionManager` zur Verfügung, der sowohl `RoleVoter` als auch `AuthenticatedVoter` nutzt.

### 9.2.3 Spring Security und Spring Web MVC

(...)