Spring Security

Seminararbeit Websecurity

Raphael Marques

4. Juni 2014

Inhaltsverzeichnis

1 Inhalt der Arbeit

Spring ist ein weit verbreitetes J2EE Framework, das vor allem im komplexeren Umfeld eingesetzt wird. Viele grosse Unternehmen setzen auf das Framework. Um eine sichere Website zu erstellen, gibt es ein Add-On namens Spring Security, welches viele Funktionen im Bereich Sicherheit, Authentisierung und Autorisierung mitbringt. Dieses Add-On wird in dieser Arbeit untersucht, erklärt sowie implementiert in einer konkreten Applikation. Verschiedene Use-Cases werden dabei getestet und mit einem Penetrationtest auf die Wirksamkeit geprüft.

1.1 Motivation

Als Sicherheitsexperte ist der Autor dieses Dokumentes sehr interessiert in standardisierten Sicherheitsframeworks und ob diese auch wirklich halten was sie versprechen. Webseiten sind im heutigem Umfeld das meist attackierte Ziel und müssen darum entsprechend geschützt werden. Eine gehackte Website richtet nicht nur einen Imageschaden oder einen Ausfall an. Sie dient auch als Malware-Verbreiter und kann als Einfallstor direkt in die Firma dienen und zum Datenklau führen.

1.2 Abgrenzung und Inhalt

Diese Arbeit beschäftigt sich mit der Konfiguration, Konzepten und praktischer Implementation von Spring Security. Das Spring Framework wird angesprochen, Erklärungen werden jedoch nur geliefert wenn nicht anders möglich.

1.3 Vorgehen

Der Autor wird sich im Detail mit dem Spring Framework, besonderem Spring Security auseinandersetzen und die Einzelnen Features und Konzepte verstehen und beschreiben. Anschliessen wird eine Implementation von Spring Security erstellt.

1.4 Zielsetzung der Arbeit

Die Arbeit besteht aus zwei Komponenten. Der erste Teil setzt sich mit dem Konzept von Spring Security auseinander und zeigt auf welche Komponenten es gibt und wie diese mit-

einander Interagieren.

Als zweiter Teil wird eine Web Applikation praktisch gesichert mit Spring Security. Folgende Richtlinien sollten dabei erfüllt werden:

- Eine sichere Implementation der Applikation mit dem Spring Framework.
- Es sollten möglichst viele Spring-Built-in Sicherheitsfeatures verwendet werden.
- Eine Dokumenation die Begründet warum welche Sicherheitsfeatures verwendet wurden und wie die verwendeten Sicherheitsfeatures funktionieren.
- Einen Penetration Test der ansatzweise "Beweisenßollte, dass das System relativ sicher ist.

2 Definition und Konzepte

Spring Security ist ein Sicherheit-Framework, welches sich in den Workflow des Spring J2EE Framework einfügt, kann jedoch auch für andere Applikationen und Frameworks benutzt werden. Dieses Dokument beschränkt sich jedoch auf den Einsatz mit dem Spring Framework. Nahezu jede Komponente innerhalb des Frameworks kann man für den eigenen Gebrauch anpassen bzw. optimieren. Dadurch kann Spring Security mit sehr vielen anderen Authentisierungs-Komponenten zusammenarbeiten (z.B. SSO Authentications, eine grosse Vielfalt von Datenbanken usw...).

2.1 Architektur

Wenn ein HTTP-Request den Webserver trifft, kommt er zuerst in die sogenannte Filter-Chain. Diese Filter-Chain basiert auf den standard HTTP Servlet filter und ist der Eingangspunkt in die Applikation. Durch die Filter-Chain wird der Request an einen sogenannten „Security Interceptor" weitergeleitet. Der Security Interceptor authentisiert und autorisiert Requests auf Basis der Konfiguration und stellt Spring Informationen über die Authentisierung bereit.

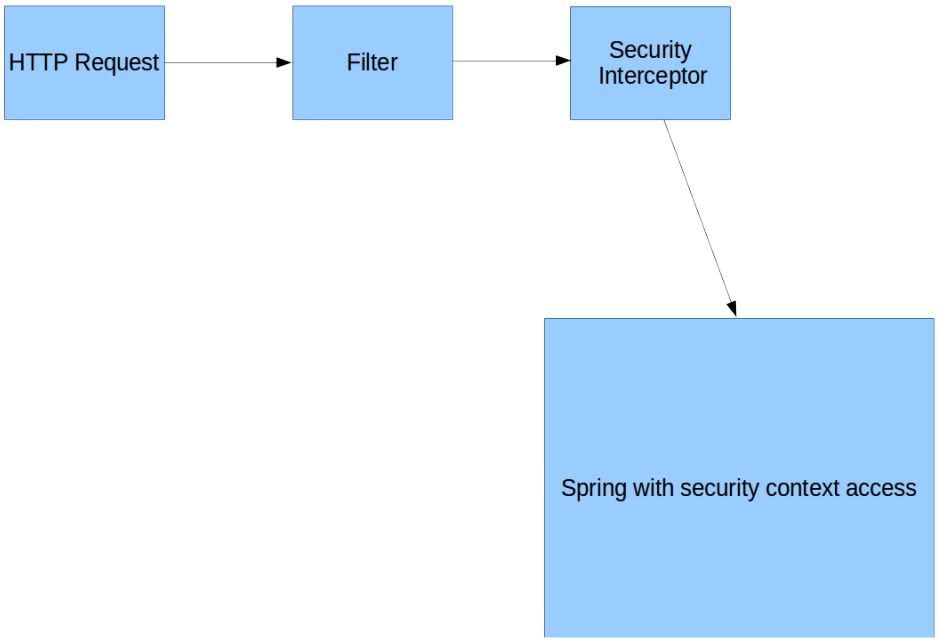

Abbildung 2.1: Übersicht Flow Spring Security,

2.2 Filter Chain

Die Filterchain basiert vollständig auf dem Java Servlet Standard. Sie entscheidet ob Spring Security anwendbar ist für eine speizifsche URL oder nicht.

Es können mehrere Filter mit der Klasse FilterChainProxy benutzt werden, welche in einer bestimmten Reihenfolge abgearbeitet werden. Es ist empfohlen diese Klasse mit Spring Security zu verwenden. Dieser FilterchainProxy kann im Web.xml konfiguriert werden.

Es können alle Java Servlet Filter oder Spring Security spezifische Filter verwendet werden. Hier ein Paar Beispiele, welche interessant für Spring Security sind:

- ChannelProcessingFilter: Evaluiert welches Protokoll verwendet wird. Kann dazu benutzt werden, um HTTP Requests an HTTPS Requests weiterzuleiten
- SecurityContextPersistenceFilter: Authentisiert schon vorhandene Sessionen, welche per JSESSION oder Cookie auf dem CLient gespeichert wurden
- UsernamePasswortAuthenticationFilter, CasAuthenticationFilter, BasicAuthenticationFilter: Fängt direkte Authentication Requests ab, wie z.B. die BasicHTTPAuthentication.
- RememberMeAuthenticationFilter: wird verwendet um einen User zu Erinnern, d.h. es wird eine Authentication generiert, ohne wirkliche Authentisierung. Dies kann nützlich sein um Userbezogenen Inhalt auf öffentlichen Seiten zu benutzen.
- ExceptionTranslationFilter Wird benutzt falls der AuthenticatoinEntryPoint einen Fehler zurückgibt (z.B. 403 Access Denied)
- FilterSecurityInterceptor Um URLS zu schützen und Exceptions zu werfen, falls der Access Denied ist.

2.3 Spring Security Konzepte - der Security Interceptor

Spring benutzt verschiedene Komponenten um Benutzer zu authentisieren und zu autorisieren. Viele dieser Komponenten können benutzerdefiniert neu implementiert werden.

2.3.1 Core Klassen

Die Core Klassen dienen als Schnittstelle mit der Applikation und den Authentication und Autorisation Komponenten innerhalb von Springsecurity. Hier werden die eigentlichen sicherheitsrelevanten Informationen über die Anfrage abgespeichert (Authentication) und Abgerufen (Autorisation und Spring)

2.3.1.1 SecurityContextHolder

Der SecurityContextHolder besitzt alle Informationen über den Security Context für den jeweiligen Request. Innerhalb von Spring kann man über dieses Objekt alle Informationen über den User herausfinden. Zum Beispiel wird hier beschrieben wie der Username heisst, in welche Gruppe er sich befindet usw.

Dieses Objekt kann nicht benutzerdefiniert neu implementiert werden.

Hier ein Beispiel wie man über den SecurityContextHolder den Username herausfindig machen kann.

```
Object principal = SecurityContextHolder.getContext()
        .getAuthentication().getPrincipal();

if (principal instanceof UserDetails) {
  String username = ((UserDetails)principal).getUsername();
} else {
  String username = principal.toString();
}
```

[2]

2.3.1.2 UserDetailService

Der UserDetailService ist ein wichtiges Core-Interface in Spring Security. Es kann beliebig implementiert werden oder es können standart Implementationen von Spring Security gebraucht werden. Der UserDetailService dient als Bindeglied zwischen Spring Security und dem User-Store. Das Interface besteht aus genau einer Methode, welche einen String braucht und ein UserDetail Objekt zurückgibt:

```
UserDetails loadUserByUsername(String username) throws
        UsernameNotFoundException;
```

[2]

Wenn die Authentisierung erfolgreich war, wird das UserDetail Objekt im SecurityContextHolder als Authentication eingepflegt.

2.3.1.3 GrantedAuthority

Im SecurityContextHolder wird neben den UserDetails auch eine Autorität gespeichert. Man kann die Autorität mit getAuthority() aufrufen. Diese Autorität definiert Berechtigungen, welche der Benutzer hat. Normalerweise sind das Gruppen oder Rollen innerhalb der Applikation. Zum Beispiel wäre die Rolle Ädminünd Üser", während die Admin Gruppe Berechtigungen zur Verwaltung der Applikation hat, hat der Benutzer die Berechtigung die Applikation zu benutzen. Spring Security kann aufgrund dieser Autoritäten die Entscheidung treffen, ob die Anfrage berechtigt ist oder nicht.

2.3.2 Authentication

Die Authentication Komponenten verifizieren die empfangenen Login-Informationen und erstellen eine Session. Die Securityinformationen werden anschliessend in einem Principal (siehe Core Components) abgespeichert und restlichen Komponenten in Spring Security sowie der Applikation zur Verfügung gestellt.

2.3.2.1 Was heisst Authentication in Spring?

Bevor auf die verschiedenen Komponenten rund um Spring eingegangen werden kann, muss zuerst genau beschrieben werden, was Authentication im Kontext von Spring Security heisst.

Die Authentisierung beinhaltet folgenden Tätigkeiten:

- Der Benutzer ruft eine Webseite auf.
- Spring Security entscheidet ob die Webseite geschützt ist oder nicht.
- Da der Benutzer noch nicht authentisiert ist, fragt Spring Security den Benutzer sich zu authentisieren
- Der Benutzer bekommt eine Möglichkeit seine Sicherheitsdetails einzugeben. (Dies Kann eine Username/Password Kombination sein (POST oder GET) , ein Zertifikat, eine Session oder etwas anderes)
- Spring Security verifiziert die Korrektheit der bereitgestellten Informationen. (z.B. Spring Security prüft die gesendete Username/Password Kombination die gleiche ist, wie in der Datenbank abgelegt)
- Falls die Informationen nicht korrekt sind, sendet Spring Security eine Access Denied Meldung zurück (Return Code 403)
- Der Kontext des Users wird abgerufen. (z.b. die Rollen, Gruppen, Berechtigungen ...)

Alle Tätigkeiten danach werden nicht mehr zu Authentication zugerechnet, sondern zu Authorisation.

2.3.2.2 AuthenticationEntryPoint

Der AuthenticationEntryPoint ist für die ersten drei Schritte zuständig. Er stellt somit den Eingangspunkt von anfragen an. Er kennt die Strategie, wohin weitergeleitet werden muss für noch nicht authentisierte User. Der AuthenticationEntryPoint kann man zusätzlich selber Implementieren oder die Default-Implementation nutzen und diesen konfigurieren.

2.3.2.3 ExceptionTranslationFilter

Access Denied Entscheidungen sind für Spring Security eine Exception. Der ExceptionTranslationFilter ist darum Verantwortlich, Access Denied Meldungen von dem AbstractSecurityInterceptor Interface (siehe Kapitel Autorisation) zu übernehmen und eine Access Denied Meldung zu generieren (Schritt 6).

2.3.2.4 UsernamePasswordAuthenticationToken

Im vierten Schritt wird ein UsernamePasswordAuthenticationToken erstellt, welches die Sicherheitsdetails beinhaltet. Dieses Token wird weiter zum AuthenticationManager gesendet.

2.3.2.5 AuthenticationManager

Der AuthenticationManager checkt ob das usernamePasswordAuthenticationToken valid ist (Schritt 5) und erstellt daraus ein Authentication Objekt (Schritt 7) (siehe unter dem Kapitel Core). Dieses Authentication Objekt beinhaltet anschliessen alles nötige für eine korrekte Authentisierung.

2.3.2.6 SecurityContext zwischen Sessions

Eine weitere wichtige Funktionalität ist das speichern des Users innerhalb einer Session. Der User sollte sich nicht jedes Mal neu authentisieren müssen wenn er auf einen anderen Link klickt. Auch dieses Feature kann man frei definieren. Als Standard wird jedoch ein sogenanntes „JSESSIONID“-Attribut mitgegeben, welches der Browser zurücksendet. Diese Session wird im SecurityContextPersistance filter abgespeichert mit allen Informationen über den Authentisierten Benutzer.

2.3.3 Autorisation

Autorisation stellt eine Berechtigungskontrolle bereit. Da der User nun authentisiert ist, geht es bei der Autorisation um den Entscheid ob der User eine Berechtigung auf eine bestimmte Ressource hat. Die Klasse, welche diese Entscheidung vornimmt ist der „AccessDecisionManager “. Die Informationen, welcher der AccessDecisionManager braucht werden von dem SecurityInterceptor - ein Interface, welches wiederum implementiert werden kann - zusammengetragen. Folgendermassen wird dabei vorgegangen:

- Alle Konfigurations Attribute des Requests werden zusammengestellt
- Die Ressource wird mit der Authentisierung und den Attributen zum AccessDecisionManager übertragen, welcher anschliessend entscheidet ob die Berechtigung vorliegt.
- Die Ressource wird ausgeführt oder es wird auf sie zugegriffen.

- Der AfterInvocationManager wird ausgeführt nachdem die Ressource ausgeführt wurde.

2.3.3.1 Konfigurations-Attribute

Konfigurations-Attribute sind die ganzheitliche Konfiguration von Spring Security. Sie definieren was wo wie zugriff hat. Z.B. Welche Gruppe auf welche Ressourcen usw. Dies kann man in einer geschriebenen Konfiguration einprogrammieren oder einen eigenen SecurityInterceptor programmieren, der solche Konfigurations-Attribute definiert.

2.3.3.2 AfterInvocationManager

Der AfterInvocationManager bildet das letzte Glied in der Kette der Autorisierung. Wenn z.B. der Access bis nach dem Methodenaufruf nicht sicher herausgefunden werden kann, so ist es dem AfterInvocationmanager möglich, nach der Ausführung der Ressource, das Ergebnis nochmal zu manipulieren und z.B. den Access revoken.

3 Implementation

Die Applikation ist unter der URL https://github.com/raphirm/squashtours abrufbar.

3.1 Environment

Das Setup benutzt folgende Komponenten:

- Spring Framework 4 (last release): Web Framework welches einzelne Webseiten und eine API bereitstellt
- Spring Security 3 (last release): Sicherung des Web Frameworks
- Spring Boot 1 (last release): Minimal-Configuration Spring und Spring Security Setup. Bietet einen einfacheren einstieg in Spring und die relevanten Komponenten können im nachhinein per Java konfiguriert werden.
- MySQL 5.3: Datenbank um Nutzer und Rollen abzuspeichern

Das Ziel dieser Implementation ist, Spring Security zu implementieren und einige Features zu benutzen. Folgende Features wurden in der Implementation benutzt:

- Multi-Method authentication: Die Webseite benutzt ein Login-Formular, während die API Requests über Basic-Authentication authentisiert
- Eigene implementationen von UserDetail und UserDetailService: UserDetail wurde implementiert und interagiert direkt mit der MySQL datenbank.
- Eigene Login-, Logout- und Access Denied Seiten für die Webseite.
- Bcrypt Passwort Encryption: State-of-the-art Passwort Encryption für eine sichere Sicherung der Passwörter in der Datenbank.
- CSFR Protection für Webseite, jedoch nicht für API.

3.2 Details zur Implementation

Abbildungsverzeichnis

Literatur

[1] Jason Ferguson. *Spring Security 3*. März 2011. URL: http://www.slideshare.net/jasonferguson1/spring-security-3.

[2] Spring Inc. *Spring Security Reference v3.2.4 RELEASE*. Mai 2014. URL: http://docs.spring.io/spring-security/site/docs/3.2.4.RELEASE/reference/htmlsingle.

[3] Dimitry Noskov. *Spring Security Framework*. März 2012. URL: http://www.slideshare.net/analizator/spring-security-framework.