Introduction to Spring Security

- Authentication and Authorization
- Configuring Spring Security

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Spring Security

- **Spring Security** is a Spring subproject that lets you apply rules for **authentication** and **authorization** to Spring applications
- Spring Security was formerly known as *Acegi Security*
- As of this writing, Spring Security is at version 3.x

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Security Modules

- **ACL** - Use Access Control Lists to secure Spring beans
- **CAS** - Integrate with JASIG's Central Authentication Service
- **Configuration** - XML configuration namespace for Spring Security
- **Core** - Basic functionality
- **LDAP** - Integrate with Lightweight Directory Access Protocol
- **OpenID** - Integrate with OpenID
- **Tag Library** - Use JSP tags to secure JSPs
- **Web** - Secure Web applications

You can find more about CAS at http://www.jasig.org/cas.

You can read more about OpenID at http://openid.net.

These modules are all packaged as JAR files that you must include in your program's CLASSPATH if use the modules' functionality.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Authentication and Authorization

- **Authentication** is the act of determining a user's (principal) identity, usually using a login name and a secret password
- **Authorization** is the act of allowing or rejecting access to resources, based on the principal's identity

The familiar login screen that many applications provide is an example of authentication. Then, based on the user's identity, the web application may grant or deny access to files, JSPs and servlets.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Programmatic versus Declarative Security

- Using **programmatic security**, your code is responsible for enforcing authentication and authorization
- Using **declarative security**, Spring Security enforces the rules, based on configuration information in the deployment descriptor
- You can use both, but we will concentrate on declarative security

These are two different strategies for implementing security. In programmatic security, the program must authenticate the user and authorize access to resources. Programmatic security is quite flexible, but potentially difficult to code and hard to maintain.

Declarative security puts the burden on an administrator rather than the programmer. The administrator writes the rules for authorization and authentication and these rules are then enforced by the container. Declarative security is quite easy on the programmer, but often does not provide as fine-grained control as programmatic security.

Spring apps can use either of these techniques or a combination of them.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Security Roles

- Spring security is based on **roles** that correspond to uses of the system (e.g. customer, administrator and so forth)
- You define the roles and assign roles to resources in a configuration file
- Spring Security uses the roles to perform authorization, e.g. you can specify which URLs in a Web application are available to each roles

Before you can apply security restraints on a Spring application, you need to define roles. A role can have one or more users. You then assign access to resources by role, not by individual user. An individual user could be assigned to multiple roles.

With declarative security, once you have defined the roles and configured resources by role, your work is done -- S pring Security is responsible for enforcing the authorization and authentication.

Note that in this regard, Spring Security is similar to JEE security.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTTP and JEE Authentication Techniques

- The servlet specification provides for:
  - HTTP Basic Authentication
  - HTTP Digest Authentication
  - HTTPS Client Authentication
  - Form-based Authentication
- Spring Security supports all of these

JEE allows for four different ways that a container can authenticate users. All use underlying HTTP authentication -- you can use those techniques in pure HTML applications. The final technique is JEE specific.

We will cover BASIC and Form-based in more detail on following pages.

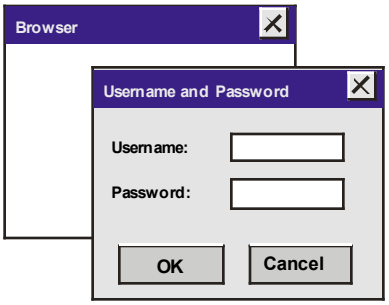
HTTP digest authentication uses strong encryption to pass the user name and password across the network. It does require that the server store actual (not hashed) passwords, so it's not appropriate for applications that want to avoid storing passwords due to security concerns.

HTTPS client authentication uses the Secure Socket Layer (SSL) and PKI client certificates.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTTP Basic Authentication

- When a user attempts to access a protected resource, the container tells the browser to display a login dialog
- You cannot control the look and feel of the login dialog
- This technique is somewhat insecure since the user name and password are passed across the network without strong encryption



The fist technique is known as "basic" HTTP authentication. When the user attempts to access a resource protected with basic authentication, the container instructs the user's browser to display a login form similar to the figure on this page. When the user submits the form, the container retrieves the username and password and authenticates them.

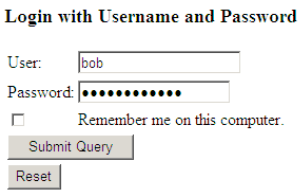
The main problem with this technique is that the browser sends the username and password over the wire in a manner that's fairly easy to decrypt. Thus, this technique is not appropriate for sensitive sites, since a cracker could intercept the login information. Note that you can use BASIC authentication over an HTTPS connection, and that IS secure, since HTTPS encrypts all traffic.

Also note that since the browser displays the login form, you are stuck with the "look and feel" provided by the browser. We will see a way to solve that issue coming up.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Form Validation

- This technique lets you control the look and feel of the login by writing HTML pages for login and error
- User name and password are passed across the network without strong encryption
- Spring Security uses this by default and provides the "custom" HTML login and error pages



This next technique is JEE-specific (not part of HTTP) -- its major benefit is that it lets you define the login and error screens instead of relying on the forms provided by the browser. That lets you provide a more consistent look and feel as well as potentially friendlier forms.

This technique suffers from the same problem as "basic" authentication in that an intruder could fairly easily intercept and decrypt the login information, but again, HTTPS is the solution for that.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Authentication Providers

- Spring Security supports many sources of credentials, including:
    - Hard-coded in XML (for development and testing)
    - Database
    - LDAP
    - OpenID

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Spring Security in Web Apps

- Spring Security lets you restrict access to URLs based on logged-in identities
- This is similar to the standard JEE Web application security, but more flexible

---

You can use Spring Security in non-Web applications, but this chapter only covers Web apps.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Steps for Using Spring Security in Web Apps

- Install the Spring Security **filter** in the application's web.xml deployment descriptor
- Include the Spring Security JAR file(s) in *WEB-INF/lib*
- Identify the URL patterns you wish to protect and the **roles** that will access them
- Write a Spring Security XML configuration file to specify URL patterns and roles

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring the Security Filter

```
1    <!-- partial web.xml contents -->
2    . . .
3
4    <filter>
5      <filter-name>springSecurityFilterChain</filter-name>
6      <filter-class>
7      org.springframework.web.filter.DelegatingFilterProxy
8      </filter-class>
9    </filter>
10
11   <filter-mapping>
12     <filter-name>springSecurityFilterChain</filter-name>
13     <url-pattern>/*</url-pattern>
14   </filter-mapping>
15
16   . . .
```


The filter provides a simple form of Aspect-Oriented Programming (AOP) - it intercepts all Web requests to the application and thus can apply security rules.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Spring Web Application Security

```
1    <!-- partial spring.xml -->
2
3    . . .
4
5    <security:http auto-config="true">
6      <security:intercept-url pattern="/index.jsp"
7        access="ROLE_ANONYMOUS" />
8      <security:intercept-url pattern="/registeredUser/*"
9        access="ROLE_USER" />
10     <security:intercept-url pattern="/admin/*"
11       access="ROLE_ADMIN" />
12   </security:http>
13
14   . . .
```


Here we configure which URLs are allowed for each role. The ROLE_ANONYMOUS is a predefined role that allows unrestricted access. Note that it's not really necessary here since the default is that URLs are not protected.

The "auto-config" attribute tells Spring Security to use several defaults for security, including "form-based" authentication (Spring Security provides the custom JSPs). You can override all of the defaults.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring The In-Memory Authentication Provider

```
1    <!-- partial spring.xml -->
2
3    . . .
4
5    <security:authentication-provider>
6      <security:user-service>
7        <security:user name="jimi" password="jimispassword"
8          authorities="ROLE_USER, ROLE_ADMIN" />
9        <security:user name="bob" password="bobspassword"
10         authorities="ROLE_USER" />
11     </security:user-service>
12   </security:authentication-provider>
13
14   . . .
```


Here we configure the users and assign their roles using the in-memory authenticator.

Note that this is not a great approach, since the password is in the clear in the XML file. It has the advantage of being easy, however. You can also use "hashed" passwords instead of storing them in the clear like this. Spring supports hashing algorithms such as MD5 and SHA, as well as bcrypt, which itself supports salted hashes.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Authentication Using a Database

- Instead of hard-coding user credentials in XML, you can configure Spring Security to query a database using the **jdbc-user-service** authentication provider
- Spring Security supports a standard schema "out of the box" or you can code custom SQL queries to match existing schema

```
-- default SQL queries

SELECT username, password, enabled
FROM   users
WHERE  username = ?

SELECT username, authority
FROM   authorities
WHERE  username = ?
```

The "authorities" table contains a mapping of usernames to the role(s) for that user.

Note that you will also need to configure a Spring DataSource bean and configure that bean to the authentication-provider bean. For example:

```
<bean id="dataSource" class="...">
  . . .
</bean>

<security:authentication-manager>
  <security:authentication-provider>
    <security:jdbc-user-service data-source-ref="dataSource" />
  </security:authentication-provider>
</security:authentication-manager>
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- Basic security concepts, including Web application security
- How to configure Spring Security

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum