Sprint Security

Spring Security

Spring Security is a powerful and highly customizable authentication and access-control framework.
It is supplied as a "ready made aspect" , from spring security framework , that can be easily plugged in spring MVC application.

It is "THE" standard for securing Spring-based applications. Spring Security is a framework that focuses on providing both authentication and authorization to Java applications.

Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements.

Features

1. Comprehensive and extensible support for both Authentication and Authorization
2. Protection against attacks like session fixation, clickjacking, cross site request forgery, etc..
3.Servlet API integration (Uses Servlet Filter chain)
4.Integration with Spring Web MVC.

Common Security Terms

Credentials : Way of confirming the identity of the user (email /username , password typically)
Principal: Currently logged in user.
Authentication: Confirming truth of credentials(i.e confirming who you are)
Authorisation: Defines access policy for the Principal.(i.e confirming your permissions, i.e what you can do)
GrantedAuthority: Permission granted to the principal.
AuthenticationManager (i/f): Controller in the authentication process.
Authenticates user details via authenticate() method.

Steps
Observation : Suddenly n automatically all end points are now protected. So on browser it will prompt you to login form (spring security supplied) On postman it will give you HTTP 401 (Un authorized error)

We have not yet supplied any credentials .
Def credentials are : user n password(UUID : universally unique ID : 128 bit) from server console.

So w/o configuring anything , the moment spring security JARs are added , all your end points are secured automatically .

Thus Spring Boot(running on the top of the Spring Framework) + Spring security , provides a ready made aspect(solution to cross cutting concern like authetication n authorization) in form of spring security

After supplying correct credentials(i.e after authentication) , spring security will redirect you to the resource : http://localhost:8080/products/view ,and you will be able to access it.
Supplies you automatically with a logout page (test it on the browser)

Observe on postman(w/o setting authorization header)
Response : (HTTP 401)

From authorization , choose Basic Authentication (referred as Basic Auth) ,
Add user name n password.

It will be encoded using base64 encoding.

Basic authentication, or “basic auth” is formally defined in the HTTP standard. When a client (your browser) connects to a web server, it sends a “WWW-Authenticate: Basic” message in the HTTP header. After that, it sends your login credentials to the server using a mild concealment technique called base64 encoding.

Not desirable , to use such credentials , so continue to next step.
---
5. Can you configure username n password n roles , in a property file ?
YES .

Add these 2 properties in application.properties file
spring.security.user.name=
spring.security.user.password=
spring.security.user.roles=

So now instead of spring security generated user name n pwd , these will be used for authentication.

---
6. Ultimate goal is using DB to store the authentication details .
BUT immediate next goal , to understand spring security is : Basic In memory authentication

The credentials will be stored in memory.
Comment earlier properties from app property file.

6.1 Add security config class.
Earlier we used to extend it from
o.s.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter
It's a convenient base class , to customize security configuration.
Read above only as Exam objective)

BUT with latest spring boot version 2.7.x onwards (Spring security 5.7.x onwards) , it has become deprecated.
So will not be used as the super class.

Ref :
https://docs.spring.io/spring-security/reference/servlet/configuration/java.html
---
6.2 Class level annotations
1. @EnableWebSecurity : for customizing spring security
2. @Configuration (annotation based approach equivalent to bean config xml file containing <bean id xml..../>) Meaning this class can declare one or more @Bean methods(for providing spring beans) and which will be auto processed by the SC . We can then inject these beans as dependencies whenever required.

6.3 Add a method , to configure a bean to provide in memory authentication , returning InMemoryUserDetailsManager

What is it ?
A class : Non-persistent implementation of UserDetailsManager which is backed by an in-memory map.
It implements UserDetailsService to provide support for username/password based authentication that is stored in memory.

InMemoryUserDetailsManager provides management of UserDetails by implementing the UserDetailsManager interface.

Mainly used for testing purpose , to be replaced by DAO based User details manager soon.

Constructor
InMemoryUserDetailsManager(UserDetails... users)

o.s.s.c.userdetails.UserDetails : i/f
Implementation class : o.s.s.c.userdetails.User
Builder Methods for configuration :
withUsername
password
roles (NOTE : NO ROLE_ prefix required here, it will be added implicitely by spring security framework)
Then build User instance.
Using : build() method
OR
Use the User ctor
User(String username,String password,Collection<? extends GrantedAuthority? authorities)

NOTE : Here in setting the authorities : ROLE_ is required.

6.4 Refer to diag : spring security architecture
Refer to readme : "spring sec auth flow"
Diagram : detailed flow.png

6.5 For supplying authorization details :
Objective :

/products/view : accessible to all
/products/add n delete : only to admin user
/products/purchase : accessible to customer role
/products/categories : accessible to any authenticated user.

Add a method , to configure a bean to provide , authorization rules.
Bean : o.s.s.web.SecurityFilterChain
Defines a filter chain which can be matched against incoming
 HttpServletRequest , in order to decide whether any filter in the chain applies to that request.

Dependency (Method Argument) : HttpSecurity

Steps
1.Authorize all requests (authorizeRequests())
2. Supply ant matcher patterns : for role specific authorization with methods like : hasRole ,hasAnyRole : no ROLE prefix or use the method : permitAll
3. Can supply the rule that remaining end points can be accessed only by authenticated users(irrespective of any role)
and :
4. Enables HTTP Basic and Form based authentication
5. Spring Security will automatically render a login page and logout success pages for you
6. Use buid() on HttpSecurity

A typical example would look like this :

```
http.authorizeRequests().
                antMatchers("/products/view").permitAll()
                .antMatchers("/products/purchase").hasRole("CUSTOMER")
                .antMatchers("/products/add").hasRole("ADMIN").
                anyRequest().authenticated().and().
                httpBasic().and().formLogin();
```

6.6 Run the application n try accessing any of the protected resource

Problem : java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "null"

Reason -- Prior to Spring Security 5.0 the default PasswordEncoder was NoOpPasswordEncoder which required plain text passwords.
From Spring Security 5, the default is DelegatingPasswordEncoder, which requires Password Storage Format.

Solution : provide Password encoder bean , in the main application class itself.

```
@Bean
        public PasswordEncoder encoder() {
                return new BCryptPasswordEncoder();
        }
```

Test the application.

6.7 Add method level finer control over authorization
1. To enable method level authorization supprt , add annotation over sec configuration class
@EnableGlobalMethodSecurity(prePostEnabled = true)
prePostEnabled : to enable pre n post sec annotations (def value is false)

2. Add pre/post annotations over controller methods
eg :
```
@PreAuthorize("hasRole('ROLE_ADMIN')")
@GetMapping("/delete")
public String deleteProduct() {
 return "Only admin should be able to delete the products";
}
```
NOTE : Method level authorization rule will override that from sec config class
---
----------In Memory authentication over----------

7. Replace in memory authentication by DB based authentication.
Using Spring Data JPA.

7.1 Edit application.properties file with DB settings.
Can optionally add these for debugging.
```
debug=true
logging.level.org.springframework.security=DEBUG
```

7.2 In Security config class
How to replace in memory authentication , by UserDetailsService based auth mgr builder
1. Simply remove in-memory configuration bean

In the absence of any specific options , default authentication provider used is : DaoAuthenticationProvider based upon UserDetailsService

(Meaning : NO extra configuration required !)

7.3 The org.springframework.security.core.userdetails.UserDetailsService interface is used to retrieve user-related data. It has one method named loadUserByUsername() which can be overridden to customize the process of finding the user.

It is used by the DaoAuthenticationProvider to load details about the user during authentication.
It is used throughout the framework as a user DAO and is the strategy used by the DaoAuthenticationProvider.

class DaoAuthenticationProvider :
Represents an AuthenticationProvider implementation that retrieves user details from a UserDetailsService.

Method
UserDetails loadUserByUsername(java.lang.String username)
                        throws UsernameNotFoundException

One important method in above i/f to implement is
        public Collection<? extends GrantedAuthority> getAuthorities() , which should return , granted authorities (role based) for the loaded user.
eg : user => UserEntity

List.of(new SimpleGrantedAuthority(user.getRole().name());

How does spring security works internally?

Spring security is enabled automatically , by just adding the spring security starter jar. But, what happens internally and how does it make our application secure?

Common Terms

Principal: Currently logged in user.
Authentication: Confirming truth of credentials.
Authorisation: Defines access policy for the Principal.
GrantedAuthority: Permission granted to the principal.
AuthenticationManager (i/f): Controller in the authentication process.
Authenticates user details via authenticate() method.

AuthenticationManager i/f implemented by : ProviderManager class .
Diagram : detailed flow .png
It Iterates an Authentication request through a list of AuthenticationProviders. AuthenticationProviders are usually tried in order until one provides a non-null response. A non-null response indicates the provider had authority to decide on the authentication request and no further providers are tried.

AuthenticationProvider: Interface that maps to a data store that stores your data.
Authentication Object: Object that is created upon authentication. It holds the login credentials. It is an internal spring security interface.
UserDetails: Data object that contains the user credentials but also the role of that user.
UserDetailsService: Collects the user credentials, authorities (roles) and build an UserDetails object.

The Spring Security Architecture

When we add the spring security starter jar, it internally adds Filter to the application. A Filter is an object that is invoked at pre-processing and post-processing of a request. It can manipulate a request or even can stop it from

reaching a servlet. There are multiple filters in spring security out of which one is the Authentication Filter, which initiates the process of authentication.

Once the request passes through the authentication filter, the credentials of the user are stored in the Authentication object. Now, what actually is responsible for authentication is AuthenticationProvider (Interface that has method authenticate()). A spring app can have multiple authentication providers, one may be using Dao based , JWT based , OAuth, LDAP ... To manage all of them, there is an AuthenticationManager.

The authentication manager finds the appropriate authentication provider by calling the supports() method of each authentication provider. The supports() method returns a boolean value. If true is returned, then the authentication manager calls its authenticate() method.

After the credentials are passed to the authentication provider, it looks for the existing user in the system by UserDetailsService. It returns a UserDetails instance which the authentication provider verifies and authenticates. If success, the Authentication object is returned with the Principal and Authorities otherwise AuthenticationException is thrown.

Spring Security Flow

1.Each incoming request passes through security filter chain for authentication and authorization process.
eg : UsernamePasswordAuthenticationFilter.
OR If the incoming request contains the authorization header "Basic base-64 encoded credentials" , this request goes through the chains of filters until it reaches BasicAuthenticationFilter.

What is the overall job of this filter?

Processes a HTTP request's Basic authorization header, putting the result into the SecurityContextHolder
---

2. AuthenticationToken is created based on User Credentials
For the user login, once the authentication request reaches the authentication filter, it will extract the username and password from the request payload. Spring security will create an Authentication object based on the username and password.

```
UsernamePasswordAuthenticationToken authentication
 = new UsernamePasswordAuthenticationToken(username, password);
```
---

3. Authentication filter delegates the request to Authentication Manager.

Authentication Manager is the core for the Spring security authentication process.

```
public interface AuthenticationManager {
    Authentication authenticate(Authentication authentication) throws
AuthenticationException;
}
```

---

4. This interface is implemented by ProviderManager class.

ProviderManager has no idea about which auth provider will support the current authentication.
So it iterates through the list of AuthenticationProviders n calls supports() method

---

5. AuthenticationProviders
The AuthenticationProvider are responsible to process the request and perform a specific authentication. It provides a mechanism for getting the user details with which we can perform authentication.

```
public interface AuthenticationProvider {

    Authentication authenticate(Authentication authentication) throws
AuthenticationException;
    boolean supports(Class<?> authentication);
}
```

eg of implementation classes :
DaoAuthenticationProvider.
RememberMeAuthenticationProvider
LdapAuthenticationProvider
...

You can also supply the custom authentication provider.

---

6. Typically used AuthenticationProvider is : DaoAuthenticationProvider
It depends upon
1. UserDetailsService
2. PasswordEncoder

6.1 Spring Security UserDetailsService

DaoAuthenticationProvider needs UserDetailsService to get the user details stored in the database by username

```
package org.springframework.security.core.userdetails;

public interface UserDetailsService {
    UserDetails loadUserByUsername(String userName/email) throws
```

UsernameNotFoundException;
}

6.2 In case of in memory based user details , UserDetailsService is implemented by InMemoryUserDetailsManager
So you can configure this as a spring bean (@Bean) in your spring sec configuration file

BUT in practical scenario , we use DB based credentials.

In that case , we will implement UserDetailsService to provide User Details
---

7. Authentication and Authentication Exception
During the authentication process, if the user authentication is successful, AuthenticationProvider will send a fully initialized Authentication object back.

For failed authentication, AuthenticationException will be thrown, filter will send the response to the client HTTP 401
For failed authorization , filter will send the response to the client HTTP 403 (forbidden)

Oherwise a fully populated authentication object carries the following details:

User credentials.
List of granted authorities (for authorization).
Authentication flag : set to true
---

8. Setting Authentication SecurityContext
The last step in the successful authentication is setting up the authentication object in the SecurityContext. It wraps the SecurityContext in the SecurityContextHolder.
---

9. The request is then delegated further to DispatcherServlet n continues with the usual flow.