# What is Spring Security?

[**Spring Security**](https://www.geeksforgeeks.org/introduction-to-spring-security-and-its-features/) is a powerful framework that provides comprehensive security features for Java applications, including **authentication, authorization**, and **protection against common vulnerabilities**.

As a key component of the **Spring Framework**, it seamlessly integrates with Spring-based projects, such as [**Spring Boot**](https://www.geeksforgeeks.org/introduction-to-spring-boot/)and [**Spring MVC**](https://www.geeksforgeeks.org/spring-mvc-framework/), to offer robust and customizable security solutions.

With built-in support for various authentication methods and advanced protection against threats like **CSRF(cross site request forgery)** and **XSS (cross-site scripting)**, Spring Security ensures your application remains secure and resilient.

In this **Spring Security Tutorial**, we'll cover the essentials of Spring Security, including setting up **authentication and authorization**, protecting your web applications from **common vulnerabilities** like **CSRF and XSS**, and integrating with systems like **LDAP(**Lightweight Directory Access Protocol **) and OAuth2**.

# Prerequisites for Spring Security

There is no prior experience in coding needed to learn Spring Security. However, if you have any prior knowledge, of below listed technologies, then it will add an advantage to your learning path.

* [Java Fundamentals](https://www.geeksforgeeks.org/introduction-to-java/)
* [Spring Framework Basics](https://www.geeksforgeeks.org/introduction-to-spring-framework/)
* [Basic of Security concepts](https://www.geeksforgeeks.org/introduction-to-spring-security-and-its-features/)

# Introduction to Spring Security and its Features

**Last Updated :**30 May, 2024

**Spring Security**is one of the projects by the Spring team that is built using the Spring framework in Java. This project aims to make it easy for developers to secure web applications against common exploits such as **Cross-Site Request Forgery** (CSRF) attacks.

It contains code that can be customized or used as-is, depending on the use case. Its major function is to manage authentication and authorization at both the Web request and method invocation levels. Internally, the spring security framework contains a series of servlet filters that handle various aspects of security. Although it adheres to Spring’s set-up conventions, programmers may select between default provisions and modify them to their specific requirements. Spring security works on the following four core concepts

* **Authentication** – Is the user really who he claims to be?
* **Authorization** – Does the user have the appropriate role?
* **Password Storage** – How is the password stored? In Memory or a database.
* **Servlet Filters**– Are there any new filters that we need to add or just use the default ones provided by the spring team?

### Advantages of Spring Security

These are some of the major advantages of Spring security.

* Protection against attacks like session fixation, csrf and clickjacking.
* Spring MVC integration.
* Support Java Configuration.
* Portable
* Integration of Servlet API
* Protect against brute force attacks.
* Active community and open source, with updates against new exploits.

### Maven Setup for Spring Security

**For setting up spring-security-core:**

You have to add the following dependency inside your [pom.xml](https://www.geeksforgeeks.org/page-object-model-pom/) file.

<properties>  
 <spring-security.version>6.0.2.RELEASE</spring-security.version>  
 <spring.version>5.2.8.RELEASE</spring.version>  
</properties>  
<dependency>   
<groupId>org.springframework.security</groupId>   
<artifactId>spring-security-core</artifactId>  
</dependency>

**For setting up spring-security-web:**

You have to add the following dependency inside your pom.xml file.

<dependencies>  
 <!-- ... other dependency elements ... -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
</dependencies>

Generally, it is considered good practice to always include spring boot security starters. It allows maven to automatically pull the supported version of [spring security](https://www.geeksforgeeks.org/spring-security-tutorial/), according to the requirements of your project. This reduces the bugs and makes version control a lot easier.

### **Spring Security Features**

1. Authorization
2. Single sign-on
3. Software Localization
4. Remember-me
5. LDAP (Lightweight Directory Access Protocol)
6. JAAS (Java Authentication and Authorization Service) LoginModule
7. Web Form Authentication
8. Digest Access Authentication
9. HTTP Authorization
10. Basic Access Authentication

**1. Authorization:**

This functionality is provided by Spring Security and allows the user to be authorized before accessing resources. It enables developers to set access controls for resources.

**2. Single sign-on:**

This feature allows a user to utilize a single account to access different apps (user name and password).

**3. Software Localization:**

This capability enables us to create user interfaces for applications in any language.

**4. Remember-me:**

With the help of HTTP Cookies, Spring Security provides this capability. It remembers the user and prevents them from logging in from the same workstation until they log out.

**5. LDAP (Lightweight Directory Access Protocol):**

That is an open application protocol for managing and interacting with dispersed directory information services over the Internet Protocol.

**6. JAAS (Java Authentication and Authorization Service) LoginModule:**

This is a Java-based Pluggable Authentication Module. It is supported by Spring Security’s authentication procedure.

**7. Web Form Authentication:**

Web forms capture and authenticate user credentials from the web browser during this procedure. While we wish to build web form authentication, Spring Security supports it.

**8. Digest Access Authentication:**

We can make the authentication procedure more secure with this functionality than with Basic Access Authentication. Before delivering sensitive data over the network, it requests that the browser verify the user’s identity.

**9. HTTP Authorization:**

Using Apache Ant paths or regular expressions, Spring provides this functionality for HTTP authorization of web request URLs.

**10. Basic Access Authentication:**

Spring Security has support for Basic Access Authentication, which is used to give a user name and password when performing network requests.

### **Features Added in Spring Security 6.0**

**1. OAuth 2.0 Login:**

This feature allows users to connect to the app using their current GitHub or Google accounts. The Authorization Code Grant defined in the OAuth 2.0 Authorization Framework is used to implement this functionality.

**2. Reactive Support:**

Spring Security 6.0 adds support for reactive programming and reactive web runtimes, as well as the ability to interact with Spring WebFlux.

**3. Modernized Password Encoding:**

Spring Security 6.0 introduces the DelegatingPasswordEncoder, a new way to store passwords. The format for storing passwords is: {id} encodedPassword. List of ids for various password encoders are:

* {bcrypt}$2a$10$dXJ3SW6G7P50lGmMkkmwe.20cQQubK3.HZWzG3YB1tlRy.fqvM/BG
* {noop}password
* {pbkdf2}5d923b44a6d129f3ddf3e3c8d29412723dcbde72445e8ef6bf3b508fbf17fa4ed4d6b99ca763d8dc
* {scrypt}$e0801$8bWJaSu2IKSn9Z9kM+TPXfOc/9bdYSrN1oD9qfVThWEwdRTnO7re7Ei+fUZRJ68k9lTyuTeUp4of4g24hHnazw==$OAOec05+bXxvuu/1qZ6NUR+xQYvYv7BeL1QxwRpY5Pc=
* {sha256}97cde38028ad898ebc02e690819fa220e88c62e0699403e94fff291cfffaf8410849f27605abcbc0

# Use Case of Spring Security

* **Authentication:**Manages user login and verifies their identity against various sources like databases or LDAP.
* **Authorization:** Controls access to different parts of the application based on user roles and permissions.
* **Protection Against Web Attacks:** Safeguards applications from common threats like cross-site scripting (XSS), cross-site request forgery (CSRF), and more.
* **Integration with Spring Ecosystem:** Seamlessly integrates with other Spring frameworks, simplifying security configuration.
* **Single Sign-On (SSO):** Supports SSO solutions, allowing users to authenticate once and access multiple applications.
* **Customization:**Allows developers to tailor security settings and behaviors to fit specific application requirements.
* **Auditing and Logging:** Tracks and logs security events for monitoring and compliance purposes.

# Spring Security - Core

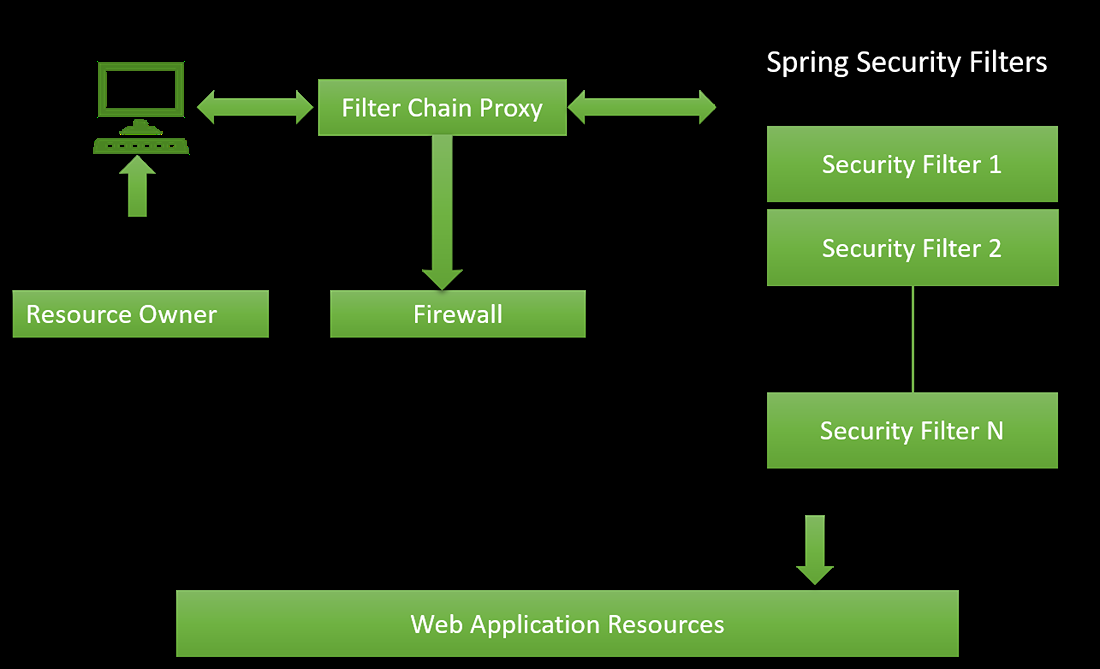
Spring Security - Core is the foundation for securing your Spring applications. This section you will get into the essential building blocks of Spring Security, including authentication mechanisms, authorization controls, and core security concepts. We'll explore how to configure user access, manage sessions, and protect against common security threats. By mastering these fundamentals, you'll be well-equipped to build robust and secure Spring applications.

# Spring Security Architecture

Spring Security framework adds two important capabilities to web applications,

1. **Authentication**
2. **Authorization/Access Control**

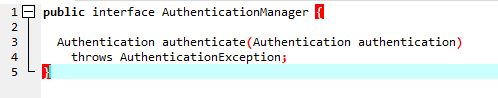
This framework provides protection against popular security issues like SCRF attacks, or Fixation attacks. It provides a secure and standard way to set up user login functionality in web applications and thus provides quick user authentication and access control.



## Authentication

Authentication is the process of verifying the identity of the computer user. It is the process of verifying the user and devices before allowing them to access the resources. In Java, the AuthenticationManager interface is responsible for handling authentication events.

**Example:**

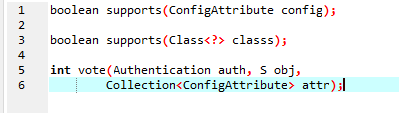


The **AuthenticationManager**interface method “authenticate()” returns authentication (i.e if authentication= true )if it verifies the identity. The **AuthenticationException**is thrown if it identifies an invalid identity or principal. It returns **null** if he cannot decide the identity.

## Authorization/Access Control

When a user or a device is authenticated, the next step is authorization which is the process of allowing the authority to perform certain tasks or operations. In Java, AccessDecisionManager and AccessDecsionVoter classes help in the authorization process.

**Example:**



The class ConfigAttribute provides the secure object metadata to provide the permission required to access it. The AccessDecisionVoter handles the Spring Expression Language (SpEL) expressions. ConfigAttribute is an interface that contains only one method that returns a string that defines the rules for access control.

## Advantages

* Provides support for Java Configuration
* Provides support for integration with Spring MVC
* Provides protection against major security issues
* Provides efficient portability

## Sample Java Configuration File

* Java

|  |
| --- |
| // Configuration Java File  import org.springframework.context.annotation.\*;  import org.springframework.security.config.annotation.web.builders.HttpSecurity;  import org.springframework.security.config.annotation.web.configuration.\*;  import org.springframework.security.core.userdetails.User;  import org.springframework.security.core.userdetails.UserDetailsService;  import org.springframework.security.provisioning.InMemoryUserDetailsManager;  import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;    @EnableWebSecurity  public class WebSecurityConfig implements WebMvcConfigurer {      // method to authenticate by collecting user related      // data      @Bean      public UserDetailsService userDetailsService()          throws Exception      {          // Managing the users in the memory database          InMemoryUserDetailsManager manager              = new InMemoryUserDetailsManager();            // Creating the new user with userid AbhijeetRathore          // and Password Abhijeet123          manager.createUser(User.withDefaultPasswordEncoder()                                 .username("AbhijeetRathore")                                 .password("Abhijeet123")                                 .roles("USER")                                 .build());            // returns User detail service          return manager;      }        // configuring MVC Configuration with HTTP Security      protected void configure(HttpSecurity http)          throws Exception      {          // define the role of the user          http.antMatcher("/")              .authorizeRequests()              .anyRequest()              .hasRole("ADMIN")              .and()              .httpBasic();      }  } |

# Spring Security Annotations

**Last Updated :**30 Apr, 2024

There are multiple annotations supported by [**Spring Security**](https://www.geeksforgeeks.org/introduction-to-spring-security-and-its-features/). But, in this article, we will discuss about these annotations can be used in a Spring Boot project as well. These annotations play a crucial role in creating a web application in Spring Boot. The Spring Security annotations are a powerful way to secure Spring applications. By using these annotations, you can control who has access to your application's resources and how they can interact with them.

## Annotations

### **1. @Secured**

* The **@Secured** annotation is used to specify a list of roles that are allowed to access a method or class. The roles are specified as a comma-separated list. **[(ROLE\_VIEWER) ,(ROLE\_ADMIN),(ROLE\_EDITOR) ]**
* The @Secured annotation doesn’t support Spring Expression Language (SpEL).

@Secured({ "ROLE\_ADMIN", "ROLE\_SUPER\_ADMIN" })

**public** void createUser(User user)

{

*// ... logic for create User*

}

### 2. @PreAuthorize

* The **@PreAuthorize** annotation is used to specify an expression that must be evaluated to be true in order for a method or class to be accessible. The expression is written in Spring Expression Language (SpEL).
* The @PreAuthorize annotation can be used to control access to methods, classes, and even entire applications.



1

@PreAuthorize("hasRole('ROLE\_ADMIN')")

2

public void deleteAdminUser(Long userId)

3

{

4

// ... logic for delete User

5

}

### 3. @PostAuthorize

* The **@PostAuthorize** annotation is used to specify an expression that must be evaluated to true after a method or class has been executed. The expression is written in SpEL.
* The @PostAuthorize annotation can be used to control the results of a method or class.



1

@PostAuthorize(

2

"hasRole('ROLE\_ADMIN') and hasPermission(returnObject, 'read:courses')")

3

public List<Course>

4

findAllCourses()

5

{

6

// ... logic for find all courses

7

}

### 4. @PreFilter

* The **@PreFilter** annotation is used to specify an expression that must be evaluated to true in order for a method or class to be included in a security filter chain. The expression is written in SpEL.



1

@PreFilter(

2

"hasRole('ROLE\_ADMIN') and hasPermission(filterObject, 'read:courses')")

3

public void

4

deleteCourses(List<Course> courses)

5

{

6

// ... logic for delete Courses

7

}

### 5. @PostFilter

* The **@PostFilter** annotation is used to specify an expression that must evaluate to true after a method or class has been executed in order for it to be included in a security filter chain.



1

@PostFilter(

2

"hasRole('ROLE\_ADMIN') and hasPermission(filterObject, 'read:classes')")

3

public List<Class>

4

findAllClasses()

5

{

6

// ...logic for find all classes

7

}

### 6. @RolesAllowed

* The **@RolesAllowed** annotation is more flexible than the @Secured annotation because it allows you to specify an expression that evaluates to a list of roles. This expression can be written in Spring Expression Language (SpEL).
* The @RolesAllowed annotation is a JSR-250 annotation, which means that it is also supported by other security frameworks, such as Apache Shiro.



1

@RolesAllowed("ROLE\_ADMIN")

2

public void deleteCourse(Long courseId)

3

{

4

// ... logic for delete course

5

}

### 7. @AuthenticationPrincipal

* The **@AuthenticationPrincipal**annotation is a Spring Security annotation that is used to inject the current authenticated user into a method or class. The user is injected as a Spring Security Authentication object.
* The @AuthenticationPrincipal annotation can be used in any Spring Security context, including web applications, RESTful APIs, and microservices.



1

@GetMapping("/username")

2

public String getUsername(@AuthenticationPrincipal String username) {

3

// ...

4

}

### 8. @RoleHierarchy

* The **@RoleHierarchy** annotation can be used to simplify the security configuration of your Spring application. By defining a role hierarchy, you can reduce the number of @RolesAllowed or @PreAuthorize annotations that you need to write.



1

@RoleHierarchy(value = { "ROLE\_ADMIN > ROLE\_USER",

2

"ROLE\_SUPER\_ADMIN > ROLE\_ADMIN" })

## Conclusion

Spring Security provides a rich set of annotations to secure Spring MVC controllers, REST endpoints, and domain model methods. So in summary, annotations provide an elegant and declarative way to implement authorization and access control in Spring Security applications.

# Spring Security with Maven

**Last Updated :**20 May, 2024

Spring Security is a powerful and highly customizable authentication access management system. This is standard for protecting Spring-based applications. Spring Security is a framework that focuses on authentication and authorization for Java applications.

Spring Security is a robust framework with breadth, designed to provide authentication and authorization capabilities for Java applications. Break down all the authentication, authorization, and security mechanisms provided by Spring Security.

## 1. Authentication

Authentication is the process of verifying that the user or system is human. It answers the question, "Who are you?" Spring Security supports several different authentication methods, including:

* In-Memory Authentication: Useful for testing or very small applications where users and their roles can be defined in application memory.
* JDBC Authentication: Uses the database to store user details and roles for the application.
* LDAP Authentication: For applications needing authentication against Lightweight Directory Access Protocol (LDAP), such as Microsoft Active Directory or Spring applications.
* OAuth2: For modern applications requiring integration with OAuth2 providers like Google, Facebook, GitHub, or other corporate authentication systems.
* Custom Authentication Providers: For unique authentication needs not covered by the built-in providers of Spring Boot.

Spring Security authentication is managed through the AuthenticationManager interface, which has a single method called authenticate(). It can be composed of one or more AuthenticationProvider instances, each of which knows how to handle specific types of authentication.

## 2. Authorization

A license determines what actions an authenticated user is allowed to perform. It addresses the question, "What are you allowed to do?" This could be simple role checks like ROLE\_USER or ROLE\_ADMIN, or more complex access control logic in Spring applications.

Spring Security handles authorization at two levels:

* Web Security: This involves configuring access to HTTP URLs. Spring Security provides options to restrict access based on URL patterns, HTTP methods, and authority checks.
* Method Security: This protects service layer operations based on annotations such as @PreAuthorize, @PostAuthorize, @Secured, and @RolesAllowed. These annotations express security rules directly in business methods.

## 3.Protection Against Attacks

Spring Security offers various protections against common vulnerabilities:

* Cross-Site Request Forgery (CSRF): It provides CSRF protection by default in web applications. It ensures that only forms originating from the application can be used to post data back.
* Session Fixation: It prevents attackers from hijacking valid user sessions during the login process of the application.
* Cross-Site Scripting (XSS): It can be handled by properly encoding data on the client side. Spring Security integrates with headers that instruct the browser to execute scripts only from trusted sources.
* CORS: CORS stands for Cross-Origin Resource Sharing. Spring Security can be configured to control which domains are allowed to access the resources, which is essential for APIs consumed by different domain frontends.

### 4. Filter Chain

The filter chain is responsible for carrying out security processes. When a request is made, it goes through a series of filters, each designed to perform a specific task. Some critical filters include:

* Authentication Filters: These filters authenticate the credentials submitted with requests.
* Management Filters: These handle various aspects of the session and other security headers.
* Access Control Filters: They decide if the request can proceed to the resources.

### 5. Configuration Models

Spring Security can be configured via XML or Java configurations. Java Configuration is type-safe, refactor-friendly, and more readable compared to XML.