Online application services are accessible anytime through desktops, laptops, tablets, cell phones and their ease of access makes them more vulnerable. There are many recent cyberattacks that happened to target various application vulnerabilities.

 The ransomware attack targeted older versions of the Windows operating environment and later fixed with relevant security patches.



 In uber attack, the hackers stole the user's personal data happened due to vulnerability in the application.



The Bangladesh bank attack, where the hackers were able to compromise Bangladesh Bank’s network. They observed how the transfers were done and gained access to the bank's credentials for payment transfers. Using these they have done a huge amount of fund transfers.



Cyberattack is increasing every year by exploiting the vulnerabilities at different levels like:

1. Operating System
2. Network
3. Application

Most of the organizations protect networks with high-end firewalls which is very difficult to penetrate and also take care of vulnerability patches for the Operating Systems. The percentage of attacks at the application level is much higher compared to the other levels.

It is essential to develop a secure application by incorporating securing measures during development processes such as secure design and coding, following security best practices and security testing, etc.

The majority of the Cyberattacks target web applications in an organization rather than its network. Attacking a network is tedious for any hacker as most of the organizations have a firewall in place which is very difficult to penetrate. Hence, it is important to gain an understanding of how to build secure web applications and fix vulnerabilities present in existing web applications.

The following are some of the de-facto standards to be considered for securing applications at design, coding, and testing.

**OWASP (Open Web Application Security Project)**

* It s a non-profitable organization which works in the area of web application and mobile security.
* Its objective is to make people aware of common and critical security vulnerabilities and measures to avoid those vulnerabilities.
* As of this course creation, OWASP Top 10 2017 is the latest vulnerabilities list

**CWE/SANS Top 25 Most Dangerous Programming Errors**

This Top 25 programming errors list is the most widespread and critical errors that can lead to serious vulnerabilities in the application, which are easy to find and exploit.

The Top 25 list is a tool for educating and creating awareness to help developers to prevent the most dangerous programming errors, thereby helping them to create a more secure application.

An overview of the common Java-based security frameworks for securing an application:

**JAAS (Java Authentication and Authorization Services)**

Java EE Security API for user authentication and authorization in Java applications.

JAAS has been integrated with Java Standard Edition Development Kit starting with J2SDK 1.4.

It can be used for authentication of users, to determine who is currently executing Java code

It can be used for authorization of users to ensure they have the required permissions to do the actions performed.

**Spring Security**

Spring Security is one of the popularly used frameworks for securing enterprise Java application

It focuses on two main application security areas such as authentication and authorization mechanisms.

It also provides many other features such as restricting URL access, session management, remember me, method level access, page level access, etc.

**Apache Shiro**

Apache Shiro is one more easy-to-use flexible and powerful Java security framework to perform authentication, authorization, cryptography, and session management.

It can be used to secure different kinds of applications such as standalone applications, web, mobile, and enterprise applications.

**OACC (Object Access Control)**

OACC is an advanced Application Security Framework for Java™ applications; that provides authentication and authorization services.

OACC runs on Java™ SE 7 (Java™ version 1.7.0), or higher.

OACC is open source software released under the commercial-friendly Apache License, Version 2.0 with a new API method to support token-based authentication

Spring Security provides powerful end to end security services for the Java EE based web/enterprise applications. It is one of the de-facto standards for securing Spring based applications.

Spring Security also helps us to overcome authentication and authorization related OWASP Top 10 vulnerabilities and SANS Top 25 Most Dangerous Programming Errors.

**Note:**

Securing application with Spring Security does not mean that your application is completely secure, there are many more security measures required to ensure your application is more secure such as:

* understanding the security requirements of your application
* creating secure architecture and design
* secure coding best practices
* security testing
* secure measures during deployment etc.

It is an open-source framework that is used for securing their applications in a platform-independent way.

Security is applied in a simpler way using declarative programming approach with annotation-based configuration.

Spring Security provides following core security services to your applications:

* Authentication
  + Basic authentication with default login/Http basic form
  + Authentication against database
  + Secure Password Storage
  + Authentication against LDAP (Lightweight Directory Access Protocol)
* Authorization
  + Role-based access
  + Restricting URL access
  + Method level security
  + Page-level security
* Session management
* Https channel security
* Remember me service

Spring Security also provides many sub-projects under its umbrella to support popular security standards/protocols such as OAuth, SAML, Kerberos.

Java EE API’s which are used to secure enterprise or web-based applications also helps us to address the security issues. But the problem with these API’s are they are not portable across EAR or WAR level. So, if you switch between the different server environments, you will end up with a lot of rework on the security configurations as they are platform specific.

Spring security overcomes all these problems in a platform-independent way and we can achieve it with a simple and flexible approach.

It provides features to protect against security attacks such as broken authentication, sensitive data exposure, cross-site request forgery, and attacks related to session management.

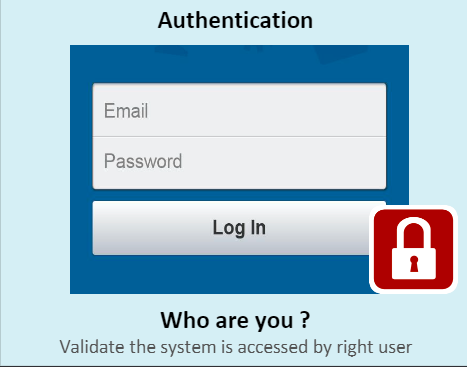
Easy to learn and apply Spring Security in an application.

Spring security is a Spring based framework, hence all the benefits of Spring are applicable to Spring Security by allowing you to develop a loosely coupled application.

Spring security addresses two main areas in application security, those are authentication and authorization areas. Let us understand more about this.

**Authentication**

It is the process of verifying the identity of the user i.e verify whether the user is the intended user or not. Example: Swipe in/out system at turnstiles at the company entrances is the best example for authentication.



**Authorization**

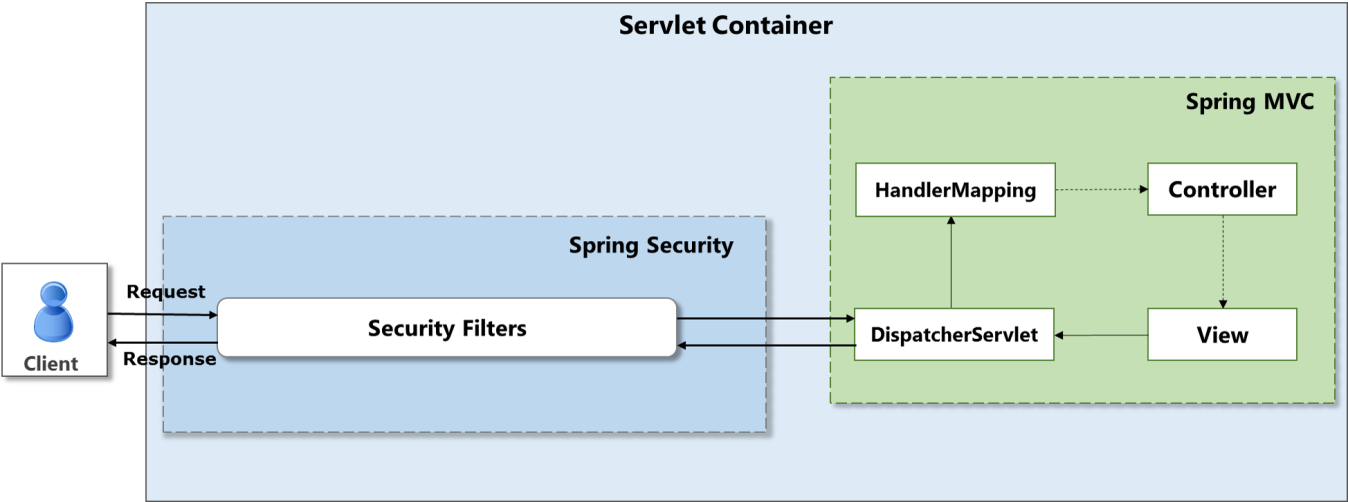
It is the process of restricting the access to system resources for authenticated users so that the person/system who is authorized to access the resource can access the same.

Example: A company can have thousands of employees, all the employees can enter the company premises but, entry to the server room is allowed only to few people.

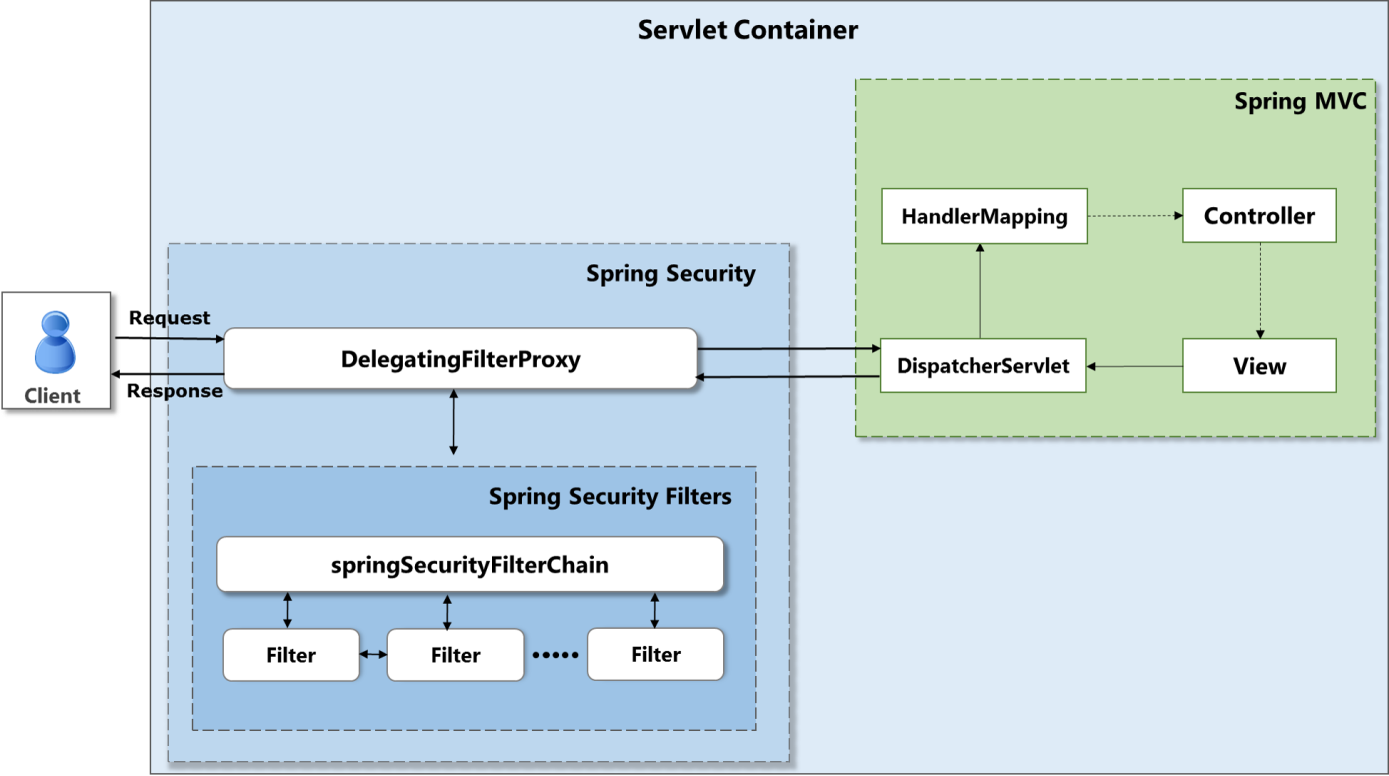


Spring Security is entirely based on standard servlet filters. Spring Security infrastructure contains a filter chain with each filter having a particular responsibility. You can add required filters through the configuration based on which security services are required for our application from Spring Security.

Spring Security filters are applied to the protected client requests before forwarding the requests to the servlet for request processing.



Look at the detailed architecture of Spring Security. You can observe the filter is a **DelegatingFilterProxy** which delegate the security request to a bean with a fixed name as**springSecurityFilterChain** from the Spring application context.  This bean contains all the required security functionalities organized internally as a chain of filters.



You need to add below given Spring Security related jars in the application classpath.

spring-security-core.jar      : Support core functionalities such as authentication, access control interfaces, and classes.

spring-security-web.jar      : Support filters and related web-security infrastructure

spring-security-config.jar   : Support security namespaces

spring-security-taglibs.jar  : Support security-related tag libraries

Maven dependencies in pom.xml to support Spring Security without Spring Boot

1. <dependency>
2. <groupId>org.springframework.security</groupId>
3. <artifactId>spring-security-core</artifactId>
4. <version>5.1.3.RELEASE</version>
5. </dependency>
6. <dependency>
7. <groupId>org.springframework.security</groupId>
8. <artifactId>spring-security-web</artifactId>
9. <version>5.1.3.RELEASE</version>
10. </dependency>
11. <dependency>
12. <groupId>org.springframework.security</groupId>
13. <artifactId>spring-security-config</artifactId>
14. <version>5.1.3.RELEASE</version>
15. </dependency>
16. <dependency>
17. <groupId>org.springframework.security</groupId>
18. <artifactId>spring-security-taglibs</artifactId>
19. <version>5.1.3.RELEASE</version>
20. </dependency>

**Note:**During this course creation, Spring Security version 5.1.3 is the stable latest version, however, you can use the latest stable version when you are working on Spring Security.

Let us look at key Spring Security configurations.

Spring Security features are applied through a chain of security filters to the specified requests before passing the requests to the servlet.

These are the two key configurations to setup Spring security to your application.

1. Setup filter chain using a subclass of **AbstractSecurityWebApplicationInitializer**
2. Creating the required Spring security configuration to apply the required security features

You can define a subclass of AbstractSecurityWebApplicationInitializer of Spring security in your application as shown below to setup the filter chain and to initialize the Security web context.

1. package com.courier.initializer;
2. import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;
3. import com.courier.config.SecurityConfig;
4. public class SecurityWebApplicationInitializer extends AbstractSecurityWebApplicationInitializer{
5. public SecurityWebApplicationInitializer() {
7. super(SecurityConfig.class);
9. }
10. }

The above class will provide a bean **springSecurityFilterChain** of **org.springframework.web.filter.DelegatingFilterProxy**class which is an internal infrastructure bean created to handle web security.

**super(SecurityConfig.class);** --> Setup required security filters in the web security context based on your application specific security features configured in Spring security configuration file **SecurityConfig.class.**

Now, let us see how to configure security features in **SecurityConfig.class.**

You can configure the required Spring security features using Java configuration.

Define a user-defined class by extending WebSecurityConfigurerAdapter as shown below.

1. package com.courier.config;
2. import org.springframework.beans.factory.annotation.Autowired;
3. import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
4. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
5. import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
6. import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
7. @EnableWebSecurity
8. public class SecurityConfig extends WebSecurityConfigurerAdapter {
9. @Autowired
10. public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
11. auth.inMemoryAuthentication()
12. .withUser("Sam").password("{noop}Sam@123").roles("USER")
13. .and()
14. .withUser("Pat").password("{noop}Pat@123").roles("EMPLOYEE");
15. }
17. @Override
18. protected void configure(HttpSecurity http) throws Exception {
19. http.authorizeRequests()
20. .antMatchers("/\*\*").hasAnyRole("EMPLOYEE","USER")
21. .anyRequest().authenticated()
22. .and().formLogin();
23. }
24. }
25. @EnableWebSecurity: This annotation enables Spring Security’s web security support to your application
26. Spring security internally uses an authentication manager which manages the authentication process with the help of an authentication provider.
    * Spring Security provides a few configuration helpers to quickly get common authentication managers to set up your application. AuthenticationManagerBuilder is one of the most commonly used helpers which is used for setting up in-memory, JDBC or LDAP user details, or for adding a custom UserDetailsService.
    * Use of @Autowired gives a bean of AuthenticationManagerBuilder to build the global AuthenticationManager
    * In this example, you are providing the user's credentials username and password as in-memory through hard coding for the authentication process as shown in the below code.

**Note:**

·         Hard coding credentials are not recommended approach, you will see how to eliminate this in the later section of this course.

·         Spring Security 5.x allows password storage only in a secure way using hashing but want to start plain text and later see secure password storage. Hence {noop} must be used along with password value in the code.

1. @Autowired
2. public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
3. auth.inMemoryAuthentication()
4. .withUser("Sam").password("{noop}Sam@123").roles("USER")
5. .and()
6. .withUser("Pat").password("{noop}Pat@123").roles("EMPLOYEE");
7. }

**configureGlobal(AuthenticationManagerBuilder auth)**only provides information about how to authenticate application users.

1. How does Spring Security know which requests to authenticate?
2. How does Spring Security know whether to support form-based or HTTP Basic authentication?
3. How to configure secure logout feature?
4. How to configure concurrent session management for restricting multiple sessions at a time?
5. How to configure URL based security which is to restrict the URL access to various roles?

All these types of required security features can be configured through the **HttpSecurity**of configure() method.

The **WebSecurityConfigurerAdapter** of Spring Securityprovides a default configuration in the configure(HttpSecurity HTTP) method that looks like:

1. protected void configure(HttpSecurity http) throws Exception {
2. http.authorizeRequests().anyRequest().authenticated()
3. .and().formLogin()
4. .and().httpBasic();
5. }

The above-given default configuration:

* Make sure that any request to the application requires the user to be authenticated
* Form-based login or HTTP Basic authentication is used to authenticate the user

However, you can also customize the configure() method based on your application requirement. Now, let us understand how the configure() method has been customized in the SwiftCourier application.

**Note:**You will learn more about form-based and http basic authentication in the **Basics Authentication** section of this course.

1. @Override
2. protected void configure(HttpSecurity http) throws Exception {
3. http.authorizeRequests().antMatchers("/\*\*").hasAnyRole("EMPLOYEE","USER")
4. .anyRequest().authenticated()
5. .and().formLogin();
6. }

In the above code, required Spring security features are applied using methods on http.

**http.authorizeRequests().antMatchers("/\*\*").hasAnyRole("EMPLOYEE","USER")**-- This code snippet ensures that only the users with role as either Employee or User can access all http requests.

**.anyRequest().authenticated()**--> This code is to say that all the requests to the application has to be authenticated

**.and().formLogin();** --> This code is to use the default Spring security login form to accept the user credentials to authenticate.

**Note:** You will learn more about role-based access control in the **Restricting URL Access**section of this course.

Core Interfaces:

Consider that you are requesting for a protected resource of your application, Spring security will intercept your request and apply configured filters to give access.

Let us look at the steps in detail.

Once you submit your authentication credentials through the browser, the authentication functionality of Spring security will collect the authentication details from the browser. Examples of authentication functionalities are form-based login and Http Basic authentication.

An authentication "**request**" object is built after collecting the authentication details and presented to the**AuthenticationManager**.

AuthenticationManager will validate the fully populated Authentication object and if the provided credentials are:

* valid then it will save the Authentication object in the **SecurityContextHolder**
* invalid then the AuthenticationManager will reject the request and will request the browser(user agent) to retry

Let us see the core interfaces of Spring Security to perform the above steps.

Core interfaces and their implementations of Spring security are

* AuthenticationManager
* UserDetailsService
* AccessDecisionManager

**AuthenticationManager**

It is an interface that performs authentication based on where the user details are stored such as in-memory, database, and LDAP, etc.

**ProviderManager**is the default implementation from Spring Security. It delegates the authentication request to a list of configured **AuthenticationProvider's**and each of which will be queried to see if it can perform authentication. The AuthenticationProvider will either throw an exception or return a fully populated Authentication object.

**UserDetailsService** : This interface helps AuthenticationProvider by giving the username, password, and authorization details either from in-memory, database, or LDAP server based on the configured source.

**AccessDecisionManager** : This interface supports in final access control decision as part of the authorization process.