Spring Security Core

Introduction

Spring Security Core refers to the essential modules and components that form the foundation of the Spring Security framework. These core components handle the primary responsibilities of authentication, authorization, and security context management, which are the critical elements for securing applications.

Key Components of Spring Security Core:

1. **SecurityContext:**
   - The SecurityContext is a central component in Spring Security that holds the authentication information of the current user. It is stored in a thread-local variable, meaning it is specific to the current thread and usually populated by Spring Security filters during the authentication process.
2. **Authentication:**
   - The Authentication interface represents the principal (usually the user) and their credentials (like a password). It also holds authorities (roles or permissions) granted to the principal. After a successful authentication process, an Authentication object is stored in the SecurityContext.
3. **GrantedAuthority:**
   - GrantedAuthority is an interface representing an authority granted to the Authentication object, typically reflecting a role or permission (like ROLE_USER or ROLE_ADMIN). These authorities are used during authorization to determine if a user has access to a specific resource or action.
4. **UserDetails and UserDetailsService:**
   - The UserDetails interface represents a user's core information, such as username, password, and authorities. It is the contract that Spring Security uses to retrieve user-related data.
   - UserDetailsService is a core interface that provides the method loadUserByUsername(String username), which is used to retrieve a UserDetails object based on a username. This is typically used in authentication processes.
5. **AuthenticationManager:**
   - AuthenticationManager is a core interface responsible for processing authentication requests. It delegates the authentication process to a list of AuthenticationProvider instances. The most

    common implementation is ProviderManager, which iterates through the configured AuthenticationProvider instances to authenticate a user.

6. **AuthenticationProvider:**
    - The AuthenticationProvider interface is responsible for authenticating a user by processing the Authentication object. Different implementations are available, such as DaoAuthenticationProvider for authentication using a database.
7. **Security Filters:**
    - Spring Security operates through a chain of filters that handle various security processes like authentication and authorization. These filters are essential in processing incoming requests and applying security logic. Key filters include:
        - UsernamePasswordAuthenticationFilter: Handles form-based authentication.
        - BasicAuthenticationFilter: Handles HTTP basic authentication.
        - BearerTokenAuthenticationFilter: Handles JWT or OAuth2 bearer token authentication.
8. **AccessDecisionManager:**
    - This interface is responsible for making final access control decisions. It evaluates whether a user has the necessary permissions to access a resource, based on the Authentication object and the resource's configuration.
9. **SecurityContextHolder:**
    - SecurityContextHolder is a helper class that provides access to the SecurityContext. It allows you to retrieve or set the current security context, typically holding the Authentication object for the current session.

How Spring Security Core Works:

- **Authentication Process:** When a user attempts to log in, Spring Security core components handle the authentication request. The AuthenticationManager (usually through its default ProviderManager) checks the credentials using an AuthenticationProvider (e.g., DaoAuthenticationProvider). If successful, an Authentication object is created and stored in the SecurityContext via SecurityContextHolder.
- **Authorization Process:** Once authenticated, the GrantedAuthority objects within the Authentication object are used to determine what

actions or resources the user can access. This is done by the AccessDecisionManager, which evaluates the user's roles against the resource's required permissions.

- **SecurityContext Management:** Throughout the user's interaction with the application, the SecurityContext is maintained by Spring Security filters. This ensures that each request is associated with the correct user authentication information.