Programming in Java

9. Spring Security

Security Concepts

- Robust software has the ability
  - “to cope with errors during execution and to handle erroneous input"
- Three types of robustness
  - Safe: when the system can detect, respond to or prevent accidental harm
  - Secure: when the system can detect, respond to or prevent intentional harm
  - Survivable: when the system is both safe and secure
- Software Engineering focuses on eliminating defects
  - Removing any faults that prevent the software from working as specified
  - Ensuring the software handles the normal and reasonable situations and inputs correctly, including invalid inputs

Security Concepts

- Software Engineering does not focus on intentional attacks
  - Attacks often involve attempting to put the system into an abnormal situation or unusual state
  - Attacks often use bizarre, unreasonable and highly unusual inputs
- Security flaw
  - A defect in or a feature of the software that can be exploited by an attacker
  - A defect that is fixed for normal operations may still be a security flaw
  - Not all defects are security flaws
  - Only defects that can be exploited are security flaws

Security Concepts

- Vulnerability: a set of circumstances allowing an attacker to exploit a security flaw
- A mitigation is the removal of a vulnerability either
  - By fixing the underlying security flaw; or
  - Applying a workaround to prevent attackers from accessing the security flaw
- Not all security flaws can be fixed
  - The cost of fixing the flaw may be prohibitive
  - The flaw may be complex or involve multiple components which means it may be a systemic problem and not a single defect

STRIDE Attack Definitions

- STRIDE is an acronym for categorizing attacks
  - Spoofing: Pretending to be something or someone else
  - Tampering: Unauthorized modification of anything in a system or application
  - Repudiation: Denying responsibility for something
  - Information Disclosure: Providing information to unauthorized parties
  - Denial of Service: Making system resources unavailable for use
  - Elevation of Privilege: Performing actions that are not authorized
- Microservices are potentially vulnerable to all these attacks
- One of the strongest mitigations to all forms of attack is robust authentication and authorization protocols

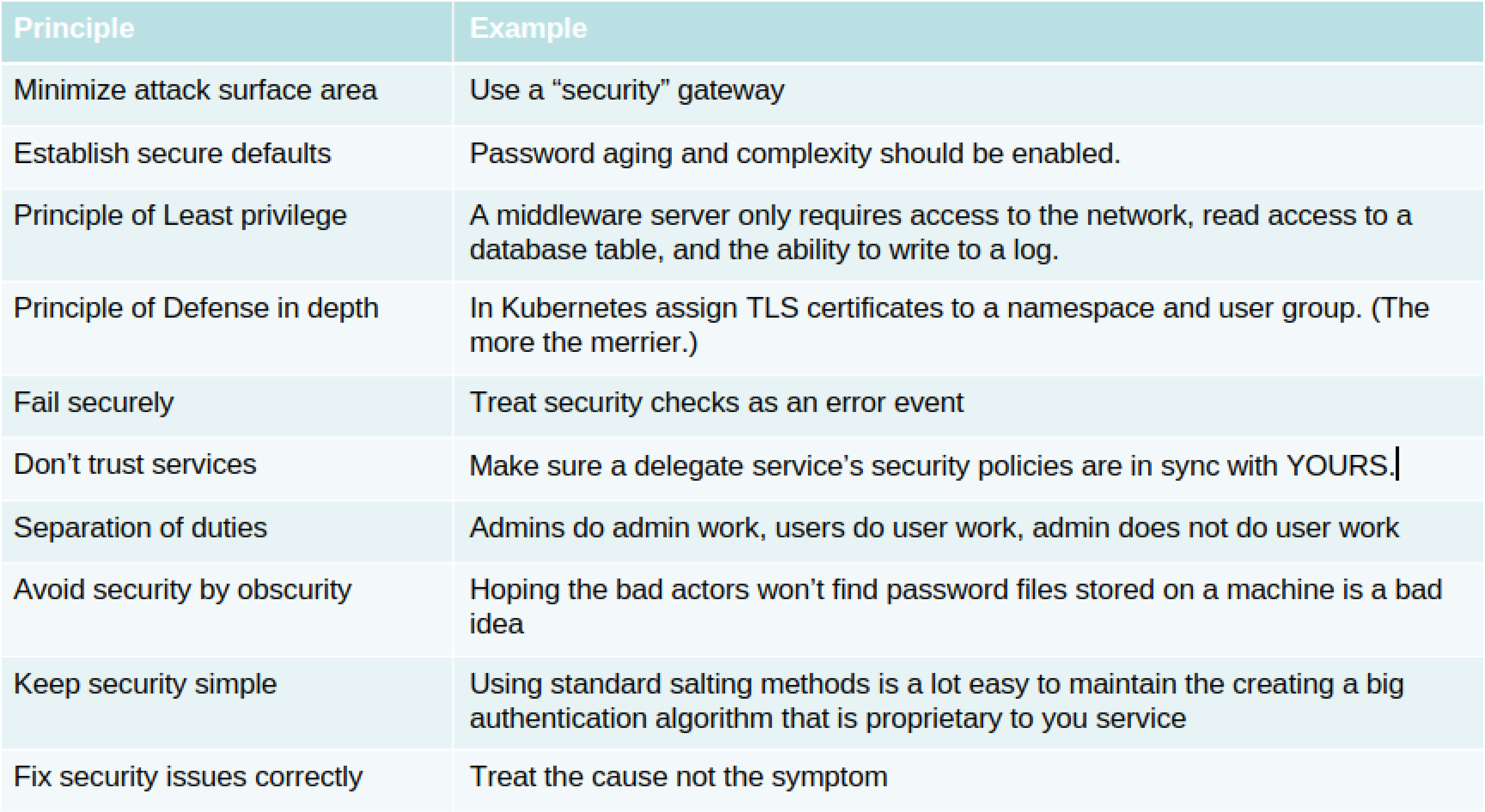
Security: Basic Principles

- Design with the objective that the API will eventually be accessible from the public internet
  - Even if there are no immediate plans to do so
- Use a common authentication and authorization pattern, preferably based on existing security components
  - Avoid creating a unique solution for each API
- Least Privilege
  - Access and authorization should be assigned to API consumers based on the minimal amount of access they need to carry out the functions required

Security: Basic Principles

- Maximize entropy (randomness) of security credentials
  - Use API Keys rather than username and passwords for API
- Balance performance with security with reference to key lifetimes and encryption/decryption overheads
- Standard secure coding practices should be integrated
- Security testing capability is incorporated into the development cycle
  - Continuous, repeatable and automated tests to find security vulnerabilities in APIs and web applications during development and testing

OWASP Secure Coding Principles



Principle	Example
Minimize attack surface area	Use a “security” gateway
Establish secure defaults	Password aging and complexity should be enabled.
Principle of Least privilege	A middleware server only requires access to the network, read access to a database table, and the ability to write to a log.
Principle of Defense in depth	In Kubernetes assign TLS certificates to a namespace and user group. (The more the merrier.)
Fail securely	Treat security checks as an error event
Don’t trust services	Make sure a delegate service’s security policies are in sync with YOURS.
Separation of duties	Admins do admin work, users do user work, admin does not do user work
Avoid security by obscurity	Hoping the bad actors won’t find password files stored on a machine is a bad idea
Keep security simple	Using standard salting methods is a lot easy to maintain the creating a big authentication algorithm that is proprietary to you service
Fix security issues correctly	Treat the cause not the symptom

Authentication and Authorization

- Authentication
  - Uses agent’s information to identify them
  - Verifies the agent’s credentials
  - Must occur before any authorization happens
  - Confirming the truth of some piece of data used by agent to identify themselves

- *“How can you prove who you are?”*

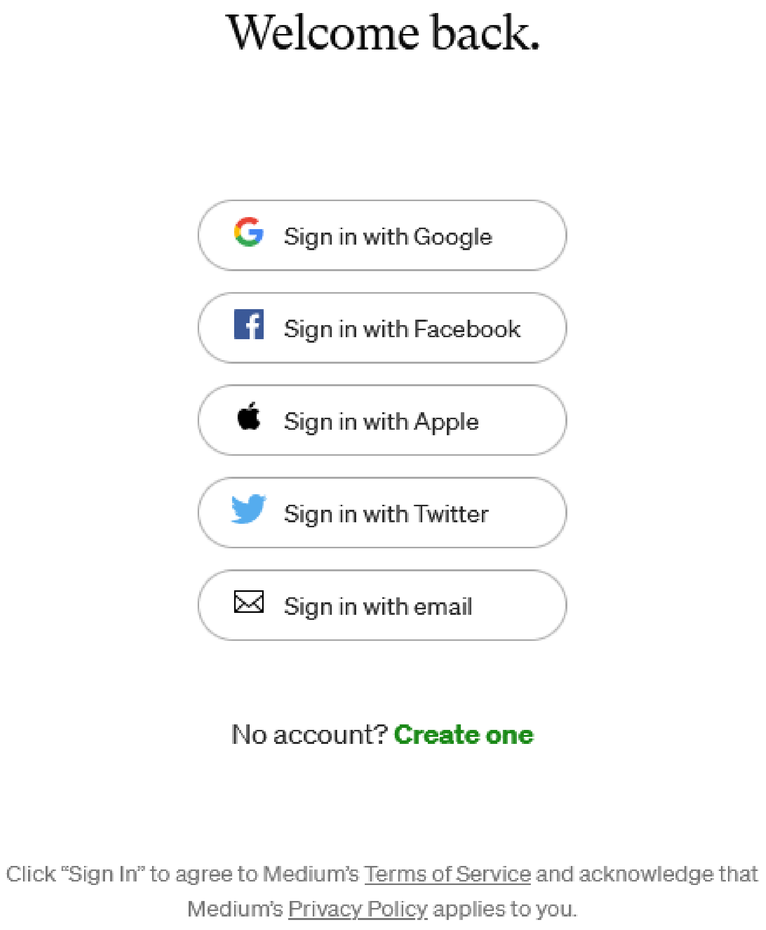
Authentication and Authorization

- Authorization
    - Checks an agent’s right to access a resource
    - Validates the agent’s permissions
    - Occurs after the identity of the agent is confirmed
    - Specifies the rights, permissions and privileges of an authenticated agent

- *“How do we know what you are allowed to do?”*

Single Sign-On

- Single Sign-On (SSO)
  - User can log in with a single ID and password to multiple systems
  - Authentication is shared between the systems
  - The systems are independent but are related in some way
  - Also referred to as a federated login across networks

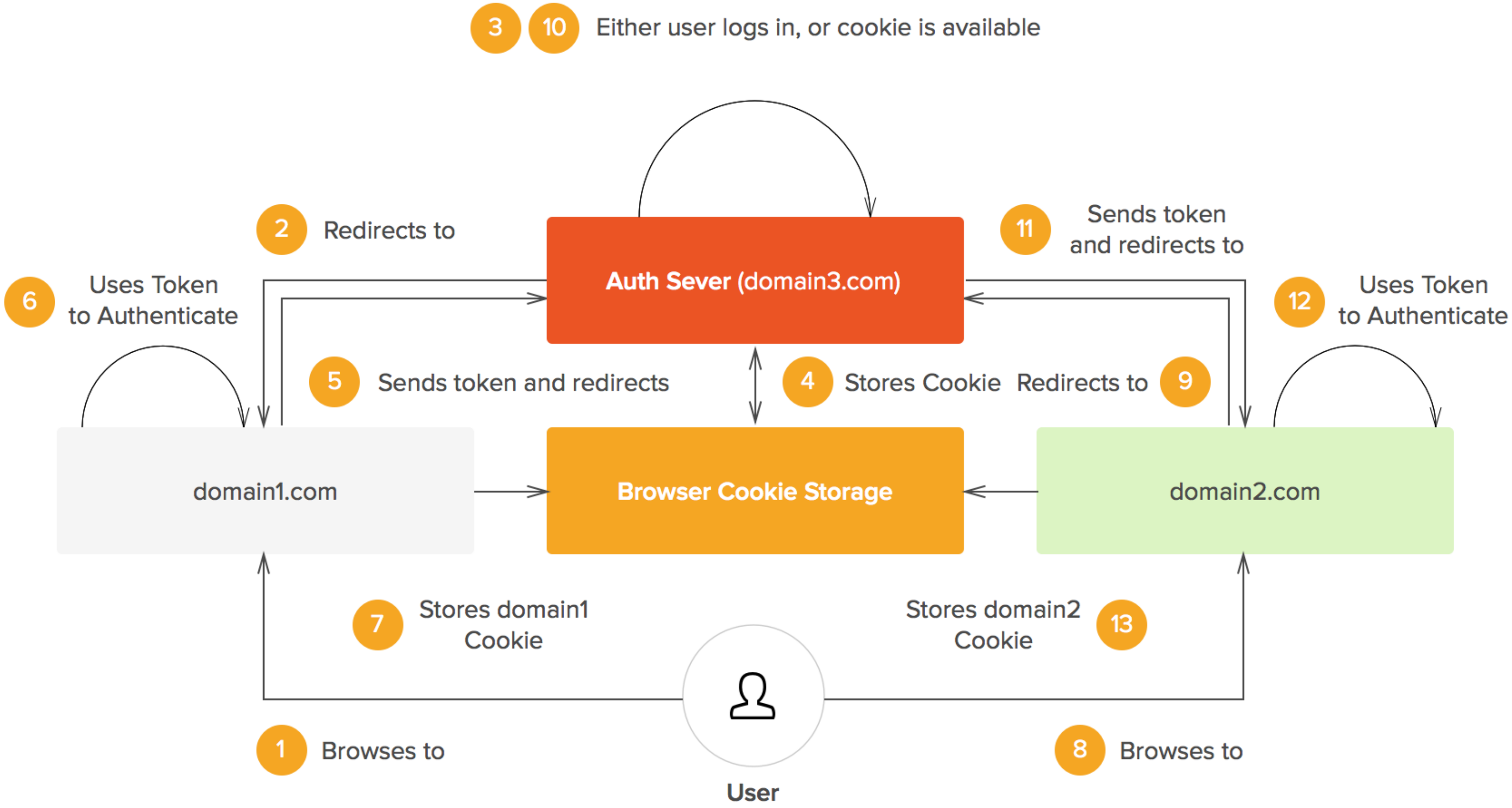


Welcome back.

No account? **Create one**

Click "Sign In" to agree to Medium's Terms of Service and acknowledge that Medium's Privacy Policy applies to you.

Identity Broker and SSO



SSO Protocols

- There exist a variety of implementations of SSO
  - Extensive list at:
    - https://en.wikipedia.org/wiki/List_of_single_sign-on_implementations
  - Some are open standards like FreeIPA from Redhat
  - Others are proprietary like Facebook Connect
- Significant challenges are:
  - How to authenticate the authenticators
  - How to communicate credentials securely
  - How to manage secrets e.g., should credentials expire?

OAuth Open Authorization

- Mechanism for providing access to a server by a client by
  - Delegating authorization to a broker which authenticates client
  - Broker returns an OAuth token used by the client to access the server
- OAuth 2 is a complete rewrite of OAuth 1
  - Not backward compatible with OAuth 1
  - The two versions are essentially separate protocols
- OAuth 2 added support for web applications, desktop applications, mobile phones, and smart devices
  - Major advantage is that devices and apps don’t store credentials
  - They only need store tokens that expire

Spring Security

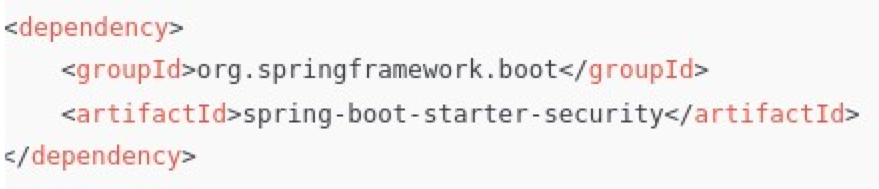
- Comprehensive, customizable authentication and access-control framework for Java applications.
    - Integrates seamlessly with Spring-based applications, enabling secure application development.
    - Protects against common attacks (CSRF, session fixation, clickjacking).
- Key Features:
    - Declarative security via annotations and configurations.
    - Pluggable authentication mechanisms.
    - Fine-grained authorization (URL-level, method-level).
    - Supports OAuth2, JWT, LDAP, and custom providers.

Core Concepts

- Authentication:
  - Verifying the identity of a user (e.g., username/password, OAuth token).
  - Managed by AuthenticationManager and UserDetailsService.
- Authorization:
  - Determining if an authenticated user has access to a resource or operation.
  - Enforced via URL-based restrictions, method-level security, and domain object security.
- Security Context:
  - Stores authentication and authorization details during a session.
  - Managed by SecurityContextHolder.
  - Available throughout the application for security checks.

Configuring Security

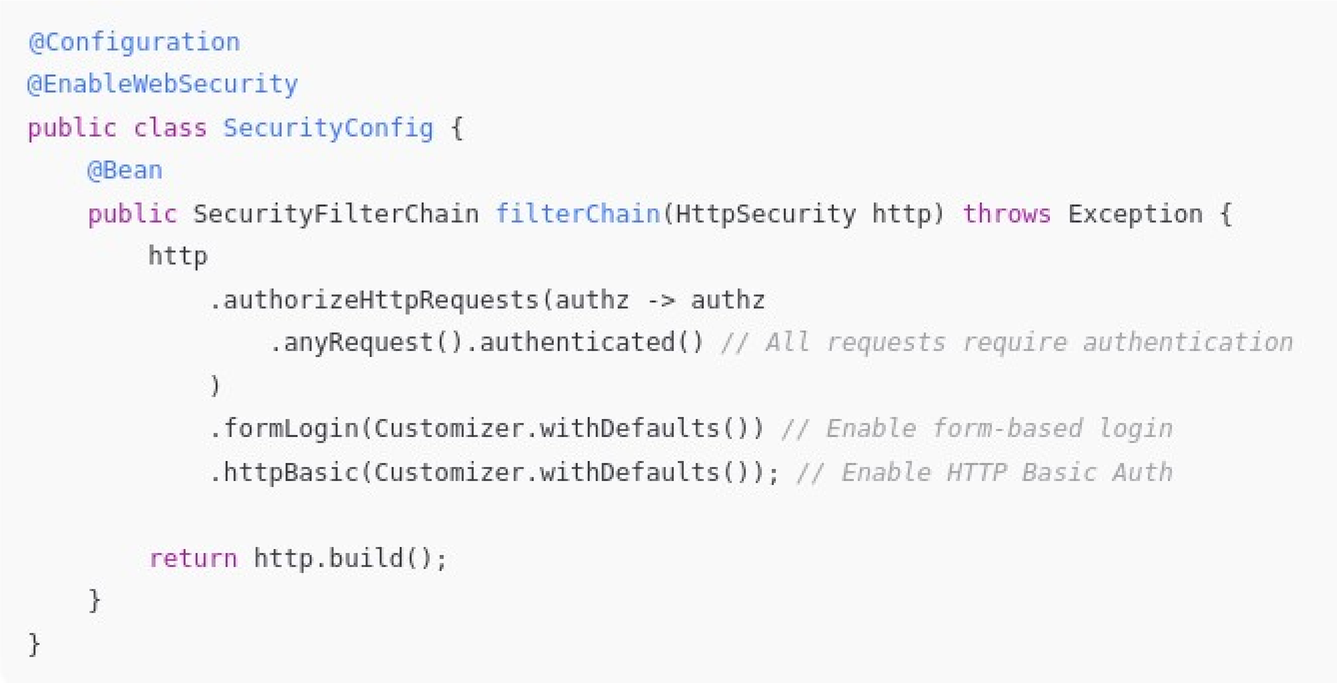
- Spring Security operates using a filter chain to process requests:
  - By default, it applies strict security (all requests require authentication).
- Configuration allows you to:
  - Define which endpoints require authentication.
  - Set up login mechanisms (form login, HTTP Basic, JWT, OAuth2).
  - Customize user details and password encoding.
  - Configure session management and exception handling.
- Spring Boot automatically adds security when the dependency is included:
- What this does by default:
  - All endpoints require authentication.
  - A default login form is provided.
  - A default user with a generated password is created on startup.



```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

SecurityFilterChain

- Declares a bean method that configures Spring Security.
  - Returns a SecurityFilterChain, which controls security for incoming HTTP requests.
  - Receives a HttpSecurity object, which is used to configure security settings for HTTP requests.
  - throws Exception because configuration methods may throw checked exceptions.

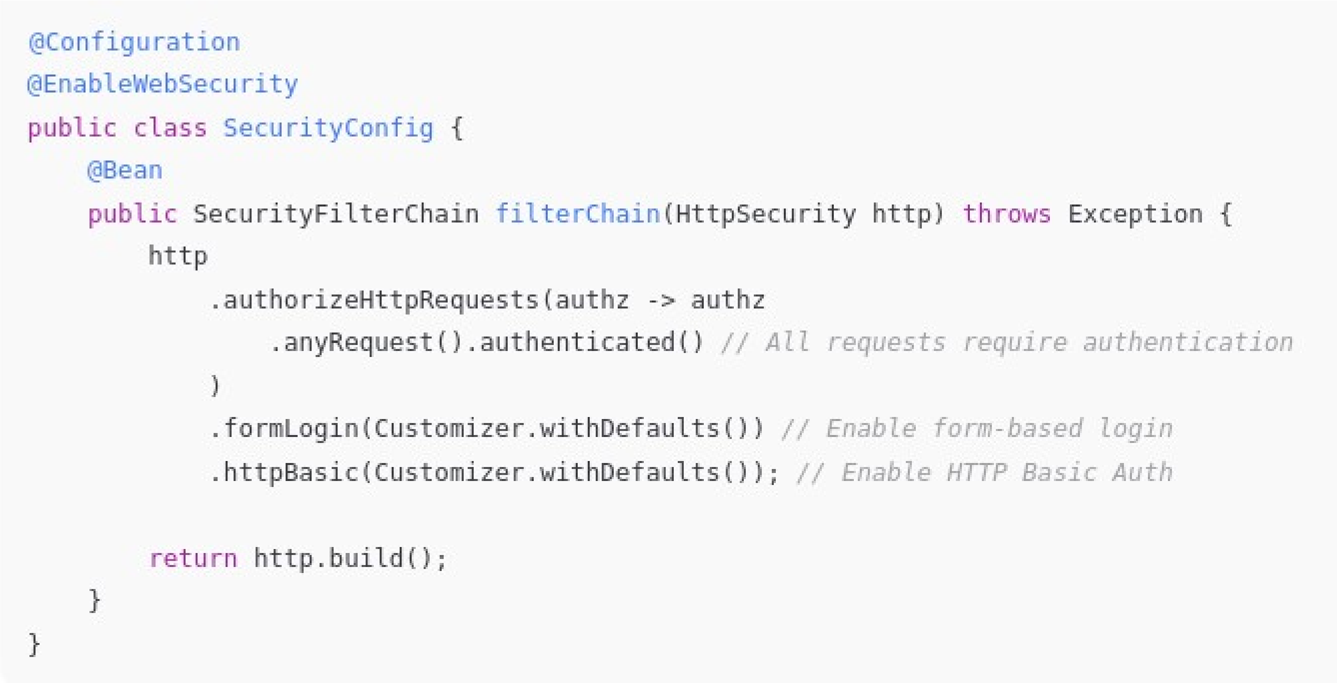


```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(authz -> authz
                .anyRequest().authenticated() // All requests require authentication
            )
            .formLogin(Customizer.withDefaults()) // Enable form-based login
            .httpBasic(Customizer.withDefaults()); // Enable HTTP Basic Auth

        return http.build();
    }
}
```

SecurityFilterChain

- Configures authorization rules for HTTP requests.
  - authorizeHttpRequests() takes a lambda to configure access rules:
    - authz -> authz.anyRequest().authenticated()
  - For any HTTP request (anyRequest()),
  - Require the user to be authenticated (authenticated()).
- Result:
  - No requests can be accessed anonymously; the user must log in first.

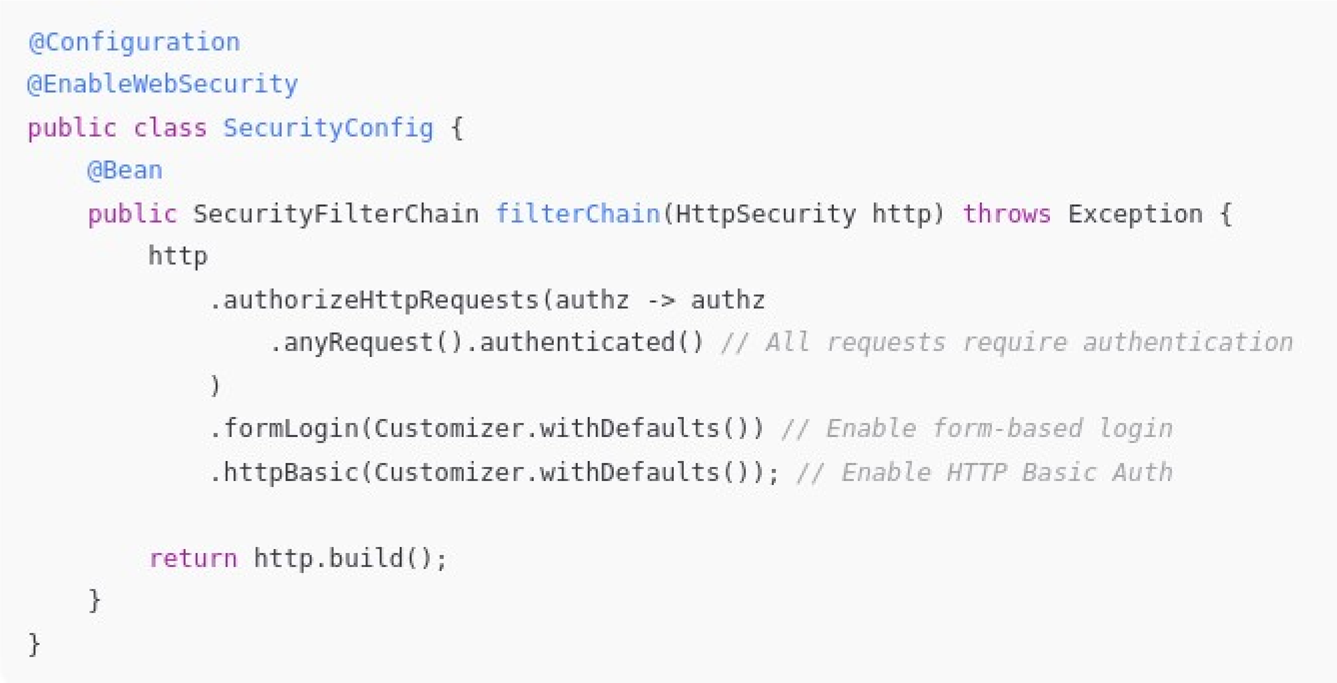


```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(authz -> authz
                .anyRequest().authenticated() // All requests require authentication
            )
            .formLogin(Customizer.withDefaults()) // Enable form-based login
            .httpBasic(Customizer.withDefaults()); // Enable HTTP Basic Auth

        return http.build();
    }
}
```

Form Based Login

- .formLogin(Customizer.withDefaults())
- Enables form-based login using Spring Security’s default login form.
- When an unauthenticated user tries to access a protected resource, they will be redirected to the login page to enter credentials.
- Customizer.withDefaults() applies default settings (default login URL /login, default success/failure handling).

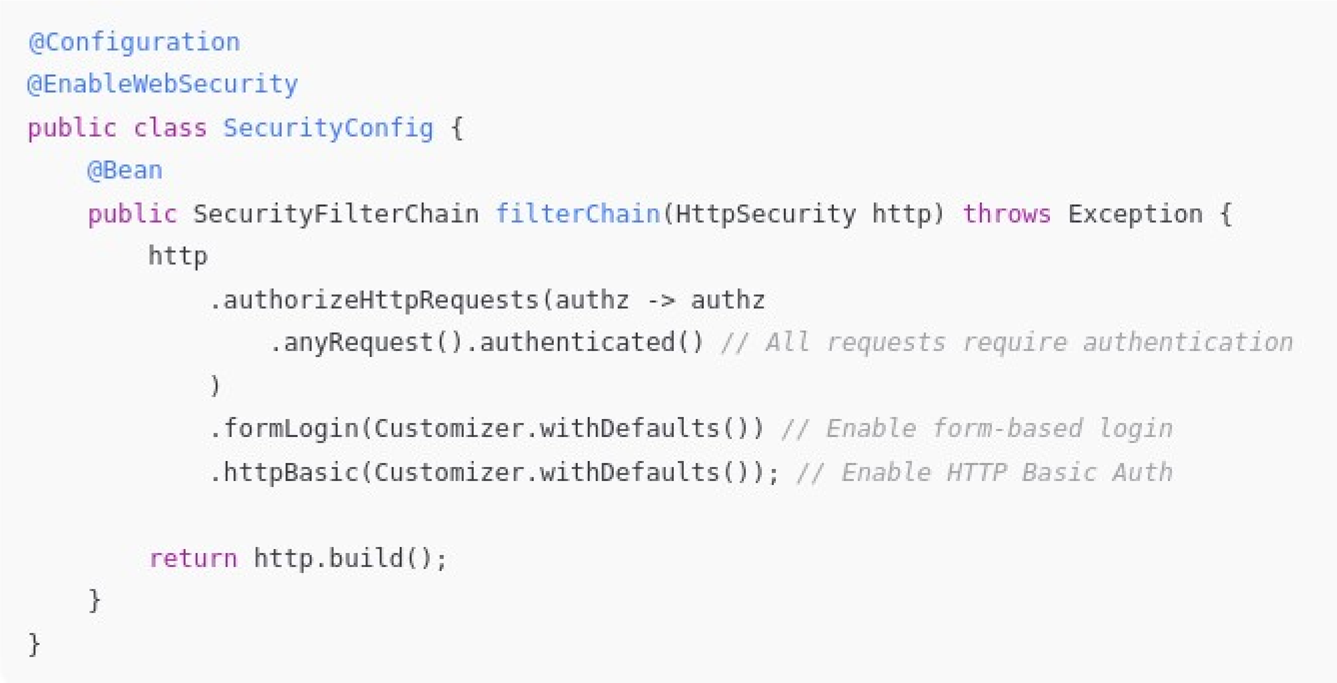


```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(authz -> authz
                .anyRequest().authenticated() // All requests require authentication
            )
            .formLogin(Customizer.withDefaults()) // Enable form-based login
            .httpBasic(Customizer.withDefaults()); // Enable HTTP Basic Auth

        return http.build();
    }
}
```

Enable HTTP Basic Authentication

  - .httpBasic(Customizer.withDefaults());
  - Enables HTTP Basic Authentication as an alternative authentication mechanism.
  - HTTP Basic prompts the browser to display a login dialog box to enter credentials.
- Useful for:
  - Simple API testing with tools like Postman/cURL.
  - Learning environments to see authentication headers.
  - Like form login, it will challenge unauthenticated requests for credentials.



```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(authz -> authz
                .anyRequest().authenticated() // All requests require authentication
            )
            .formLogin(Customizer.withDefaults()) // Enable form-based login
            .httpBasic(Customizer.withDefaults()); // Enable HTTP Basic Auth

        return http.build();
    }
}
```

Lab 9-1

Spring Security

Java™