Web Security

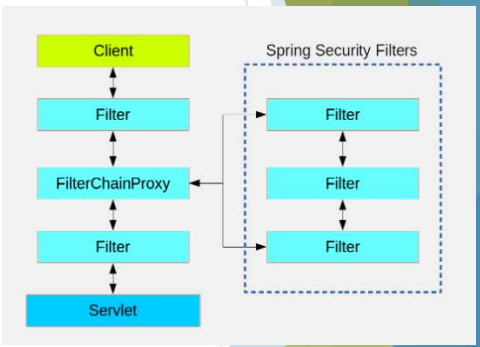
andy

1

Where Security Fits.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/spring/*.xml
        </param-value>
    </context-param>

    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```

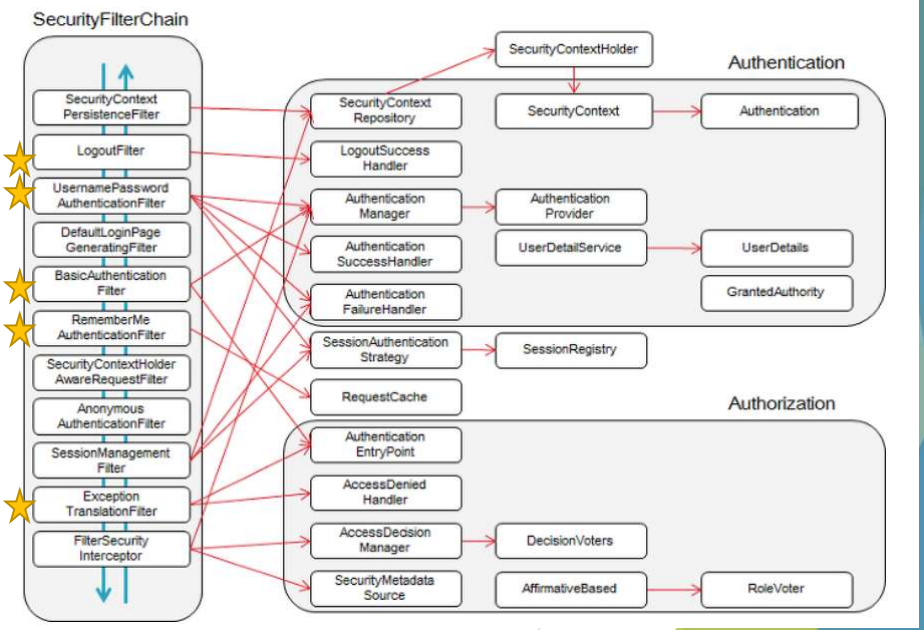
Web.xml



Security Chain.xml

2

Security Filter Chain



3

Customize the filters

- Spring security filter chain is ready to work out of box. But you always need to customize “some” filters to fit your own application.
- The Only abstract class: WebSecurityConfiguerAdapter
  - 1. Override the configure(HttpSecurity) method to customize the filter – customize authentication and access control (authorization)
  - 2. Override the configure(HttpSecurity) method to add your own filter – for filters not covered by spring built-in filter chain
  - 3. Override the configure(AuthenticaionManagerBuilder) to connect your database for securitycontext.

4

Authentication

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception{
    auth
            .userDetailsService(userDetailsService)
            .passwordEncoder(passwordEncoder()); //username password authentication

    auth.jdbcAuthentication();      //jdbc authentication
    auth.inMemoryAuthentication();  //in memory authentication

}
```

5

Authorization

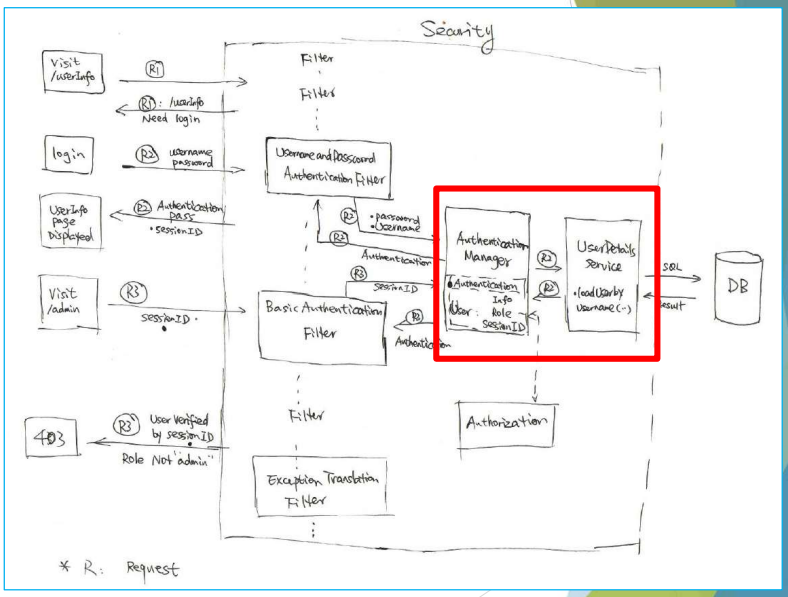
```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    .....

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http.csrf().disable();

        //Authorization - access control
        http.authorizeRequests().antMatchers("/", "/signup", "/login", "/logout").permitAll();
        http.authorizeRequests().antMatchers("/userInfo").access("hasAnyRole('USER', 'ADMIN')");
        http.authorizeRequests().antMatchers("/admin").access("hasRole('ADMIN')");

        //access denied page
        http.authorizeRequests().and().exceptionHandling().accessDeniedPage("/accessdeny");
    }
}
```

6

Login Security

Your implementation:

1. Customize UserDetailsService
2. wire to AuthenticationManager



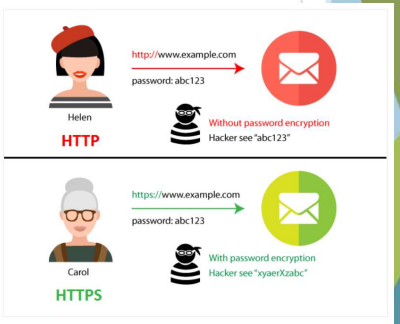
7

Password Encoding

- History:
  - Plain password
  - SHA-256
  - bcrypt

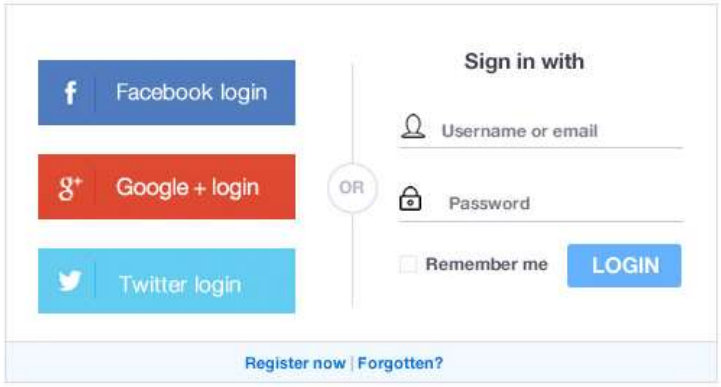
```
{bcrypt}$2a$10$dXJ3SW6G7P50lGmMkkmwe.20cQQubK3.HZWzG3YB1tlRy.fqvM/BG
{noop}password
{pbkdf2}5d923b44a6d129f3ddf3e3c8d29412723dcbde72445e8ef6bf3b508fbf17fa4ed4d6b99ca763d8dc
{scrypt}$e0801$8bWJaSu2IKSn9Z9kM+TPXfOc/9bdYSrN1oD9qfVThWEwdRTnO7re7Ei+fUZRJ68k9lTyuTeUp4of4g24hHnazw==$OAOec05+bXxvuu/1qZ6NUR+xQYvYv7BeL1QxwRpY
{sha256}97cde38028ad898ebc02e690819fa220e88c62e0699403e94fff291cfffaf8410849f27605abcbc0
```

- Http vs HTTPS
  - "s = secure" -> SSL certificate
  - Browser and application both needs to adopt SSL certificate
  - Http transfer password in PLAIN TEXT; Https transfer password in Encrypted Format



8

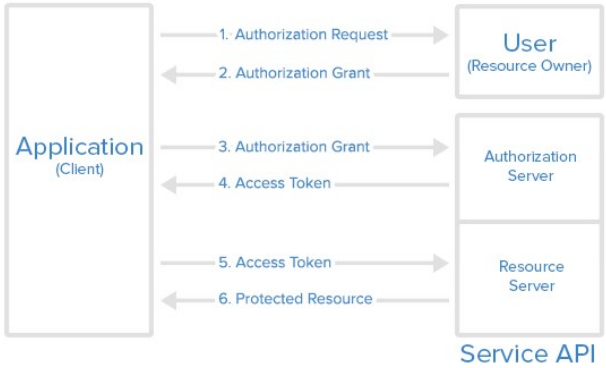
Oauth2 - SSO



9

Oauth2 workflow

Abstract Protocol Flow



Example:

"You" use "login with google" to login to your ebay.com

- Application: ebay.com
- User: You
- Authorization Server: Google
- Resoruce Server: Google

10

Implement Oauth2

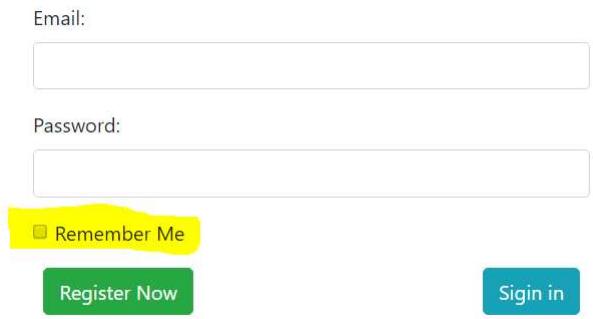
- 1. register your application in the authorization/resource agent
- 2. create oauth2 filter and add it to the security filter chain

```
@Configuration
@EnableWebSecurity
@EnableOauth2Client
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // ***** Oauth2 -- add filter to spring security filter chain  *****
        http.authorizeRequests().and().addFilterBefore(ssoFilter(), BasicAuthenticationFilter.class);

    }

    // ***** define the filter *****
    private Filter ssoFilter(){
        //create google filter
    }

    @Bean   // ***** Register filter as spring security filter *****
    public FilterRegistrationBean<OAuth2ClientContextFilter> oAuth2ClientFilterRegistration(OAuth2ClientContextFilter
    filter){
        FilterRegistrationBean<OAuth2ClientContextFilter> registrationBean = new FilterRegistrationBean<>();
        registrationBean.setFilter(filter);
        registrationBean.setOrder(-100);
        return registrationBean;
    }
}
```

11

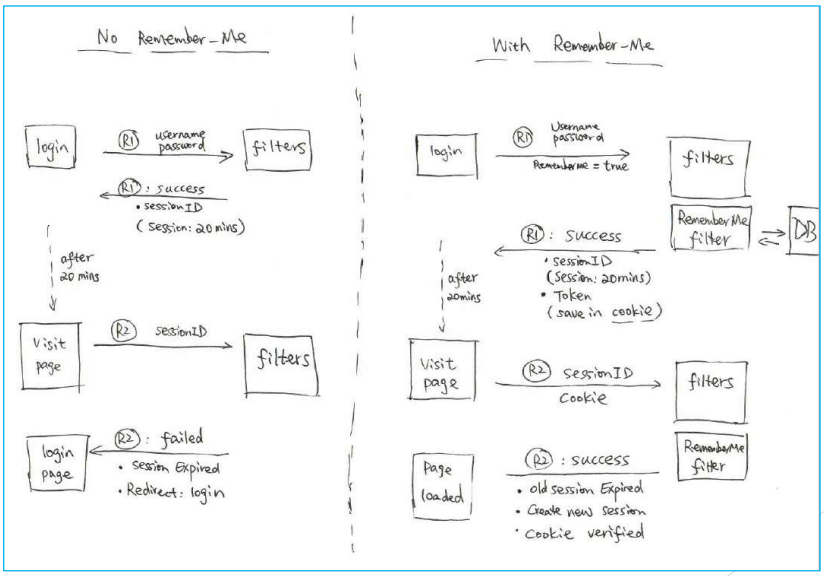
Remember Me

- Remember your credentials even when your session is over
- Remember for certain period
- Invalidate with logout.



12

Remember-Me workflow



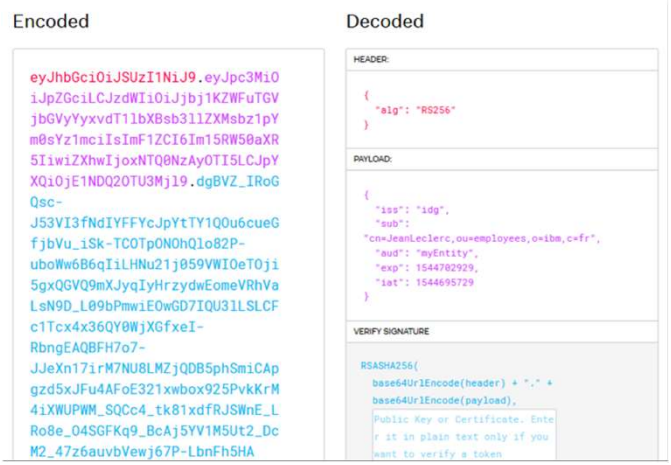
13

Implement Remember-Me

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        //Remember Me
        http.authorizeRequests().and()
                .rememberMe()
                .rememberMeParameter("remember-me")  //cookie entry name
                .tokenRepository(persistentTokenRepository())
                .tokenValiditySeconds(24 * 60 * 60);  //remember-me for one day
    }


    @Bean //will save token in database using below JdbcTokenRepositoryImpl
    public PersistentTokenRepository persistentTokenRepository(){
        JdbcTokenRepositoryImpl tokenRepositoryImpl = new JdbcTokenRepositoryImpl();
        return tokenRepositoryImpl;
    }
}
```

14

JWT token



- **Header:** algorithm (RS256).
- **Payload:** credentials Data
- **Signature:** calculated based on Header and Payload, used to verify sender information and verify payload is not changed

15

Symmetric vs Asymmetric Encryption

- Lock(for encrypt) vs Key(for decrypt)
  - Symmetric: the same lock and key is used to encrypt and decrypt the data
    - Use Case: *User1 wants to securely transfer data instead of plain text via network to User2, User1 encrypt the data with AES-256 and to decrypt the data, Key1 should be used. User2 receives the encrypted data, in order to decrypt it to be readable. User1 must share Key1 to User2.*
    - Example: AES, DES
    - Problem: User1 and User2 have to share the Key
  - Asymmetric: encrypt and decrypt using different key (**public key and private key**)
    - Use case: *User1 wants to securely transfer data instead of plain text via network to User2. User1 first find the public key from User2 (User2 give public keys to whoever ones to send secure data to him). User1 encrypt the data with the public key and send it to User2. User2 receives the data, he then use his private key to decrypt it to be readable. User1 encrypt with public key. User2 decrypt with private key.*
    - Example: RSA
    - Benefit: User2 does not have to share his private key.

16

Question

- How do you protect your REST api?
- How do you configure security in your application?
- Why is Https securer than Http?
- How do you prevent regular user visit “/admin” url?
- What is JWT token? What is it made of?
- What is public key encryption/asymmetric encryption?

17