Tema 3. Tecnologías del servidor. Seguridad: Spring Security

Programación web

Boni García

Curso 2017/2018

Índice

Índice

Introducción

Servicios de seguridad

- Un **servicio de seguridad** protege las comunicaciones de los usuarios ante determinados ataques. Los principales son:
  - Autenticación (*authentication*): sirve para garantizar que una entidad (persona o máquina) es quien dice ser
  - Autorización (*authorization*): sirve para discernir si una entidad tiene acceso a un recurso determinado
  - Integridad (*data integrity*): garantiza al receptor del mensaje que los datos recibidos coinciden exactamente con los enviados por el emisor
  - Confidencialidad (*data confidentiality*) proporciona protección para evitar que los datos sean revelados a un usuario no autorizado

Introducción

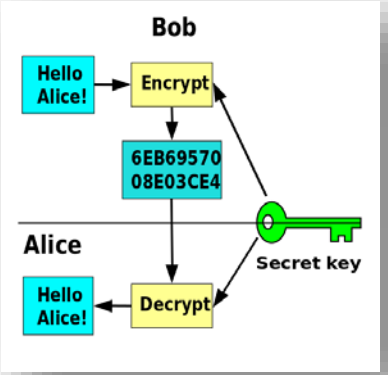
Servicios de seguridad

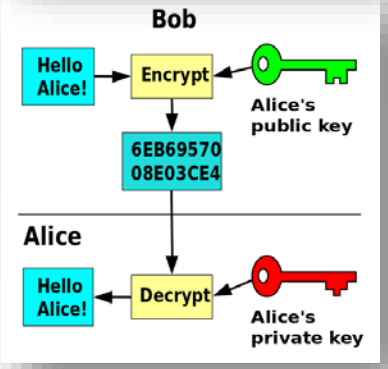
- La **autenticación** se consigue mediante:
  - Algo que sabes. Por ejemplo, unas credenciales login-password
  - Algo que tienes. Por ejemplo, una tarjeta de acceso
  - Algo que eres. Por ejemplo, cualidades biométricas (huella digital…)
- La **autorización** discrimina el acceso a un determinado recurso en base a permisos (*grants*), roles de usuario, tokens, …
  - A veces requiere autenticación previa (es decir, confirmar la identidad)
- La **integridad** se consigue típicamente con funciones Hash (resumen)
  - Son funciones computables mediante un algoritmo que convierte una entrada binaria (típicamente un fichero o un mensaje digital) a un rango de salida finito (típicamente una cadenas alfanumérica)
  - La posibilidad de colisión (diferentes entradas con mismo hash) es muy pequeña

Introducción

Servicios de seguridad

- La **confidencialidad** se consigue usando técnicas criptográficas (cifrado de mensajes). Tipos de sistemas criptográficos:
  - Criptosistemas de **clave secreta**. En ellos, la clave de cifrado y de descifrado es la misma: es una clave secreta que comparten el emisor y el receptor del mensaje. Debido a esta característica son denominados también criptosistemas **simétricos**
  - Criptosistemas de **clave pública**. Se distinguen porque cada usuario o sistema final dispone de dos claves: una privada, que debe mantener secreta, y una pública, que debe ser conocida por todas las restantes entidades que van a comunicar con ella. Se los conoce también como criptosistemas **asimétricos**





Introducción

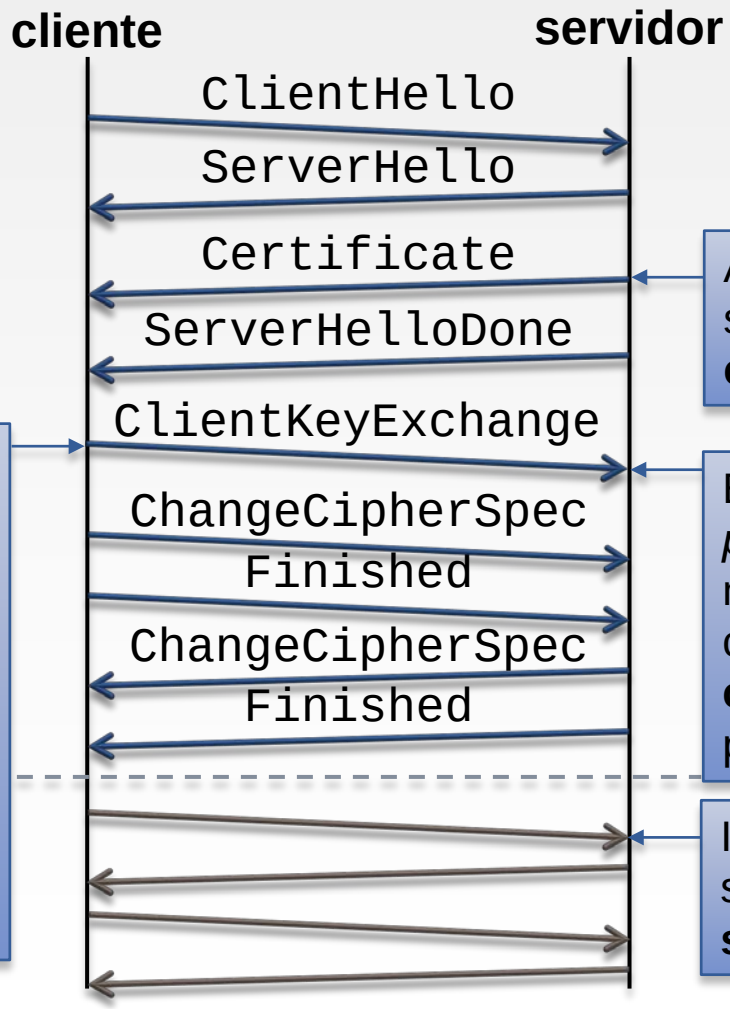
TLS

- TLS (*Transport Layer Security*) es un protocolo criptográfico de nivel de transporte que proporciona comunicaciones seguras (cifradas) a una conexión TCP
  - Es la versión evolucionada de SSL (*Secure Sockets Layer*)
  - En octubre de 2014 se descubrió una vulnerabilidad crítica en SSL 3.0 que hace que su uso esté desaconsejado
- Los servicios de seguridad ofrecidos por TLS son confidencialidad, (cifrado el intercambio de datos a nivel de transporte), autenticación (entidades pueden confirmar su identidad), e integridad (mediante una función hash)
- Para establecer un canal seguro cifrado, las entidades tienen que llegar a un acuerdo (***handshake***)

Introducción

TLS

- Handshake TLS en aplicaciones cliente-servidor (por ejemplo en HTTPS)



El cliente genera una clave *pre-master* que será usada generar la clave maestra MS con la que se cifrarán todos los datos de la sesión. Esta clave se envía cifrada con la **clave pública del servidor**, obtenida a partir del certificado

Al enviar el **certificado** el servidor está mandando su **clave pública** al cliente

El servidor obtiene la clave *pre-master* descifrando el mensaje enviado por el cliente, usando para ello **su clave privada**. A partir de la pre-master se obtiene la MS

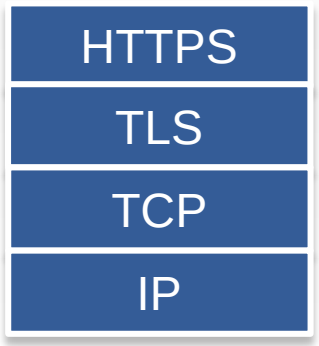
Intercambio de datos en canal seguro mediante cifrado **simétrico**

Introducción

HTTPS

- *Hypertext Transfer Protocol Secure*. Versión segura de HTTP
- HTTPS no es más que HTTP sobre TLS
- Con HTTPS se consigue que la información sensible (claves, etc.) no pueda ser interceptada por un atacante, ya que lo único que obtendrá será un flujo de datos cifrados que le resultará imposible de descifrar
- Puerto TCP por defecto en servidores HTTPS: 443

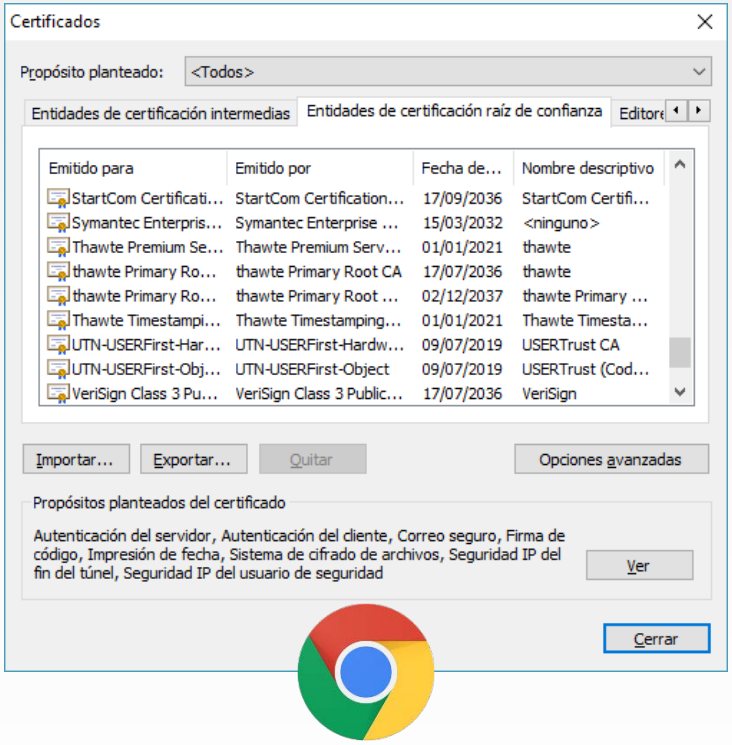
Chrome 68 (planificado para julio de 2018) marcará como “no seguras” todas las páginas web servidas sin HTTPS



Introducción

HTTPS

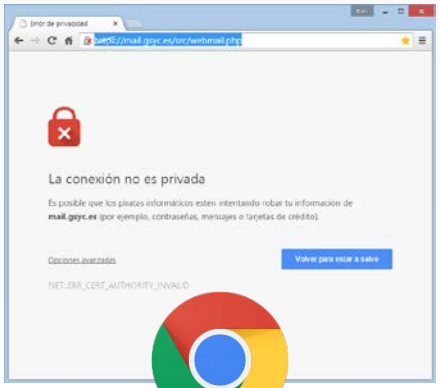
- Un **certificado** es emitidos por Autoridad de Certificación (CA)
  - Ejemplos: FNMT, GlobaSign, Symantec, Verisign, …
- Los navegadores tienen una lista de CAs conocidas
- Alternativa gratuita para obtener un certificado reconocido por los navegadores: https://letsencrypt.org/

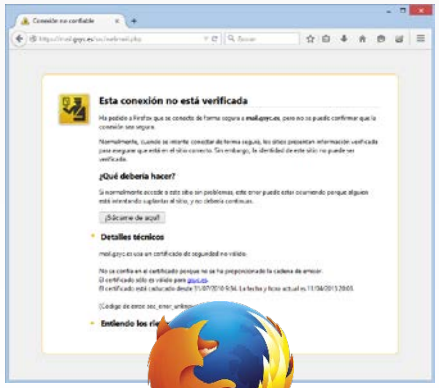


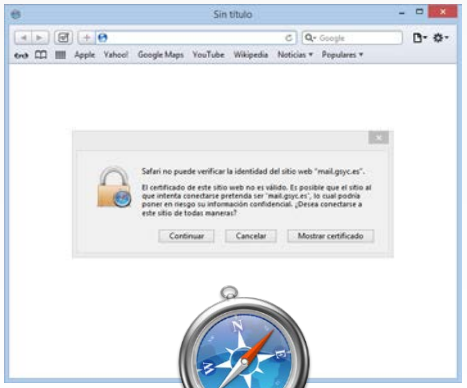
Introducción

HTTPS

- Al recibir un certificado emitido por una CA desconocida, el navegador muestra una alerta de seguridad
- Por ejemplo, al recibir un certificado autofirmado
  - Certificado firmado por la propia entidad cuya identidad se quiere autenticar
  - Se puede generar por ejemplo con OpenSSL o KeyTool (para Java)







Introducción

Vulnerabilidades web

- Algunas de las **vulnerabilidades** web más importantes son:
  - *Injection*: Método de infiltración de código malicioso, por ejemplo SQL en el lado servidor
  - *Cross-Site Scripting* (XSS): Inyección de código (típicamente JavaScript) malicioso en la lado cliente (por ejemplo para robar cookies)
  - *Cross Site Request Forgery* (CSRF): Falsificación de petición. Ocurre cuando a un usuario legítimo (típicamente con sesión en una web) se le fuerza a hacer una petición no deseada (por ejemplo cambiar el password)
- **OWASP** (*Open Web Application Security Project*) es una organización sin ánimo de lucro dedicada la seguridad de aplicaciones web
  - Tutoriales, herramienta, metodologías, …

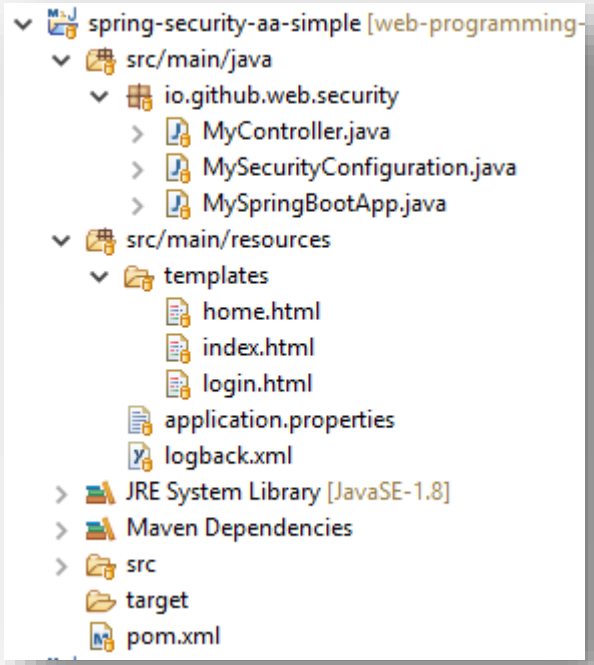
https://www.owasp.org/

Autenticación y autorización en Spring Security

- Vamos a ver como implementar los servicios de seguridad de autenticación y autorización en Spring Boot y Spring Security estudiando tres ejemplos:
  1. Sencillo (proyecto `spring-security-aa-simple`)
     - Usuarios en memoria, rol único
  2. Medio (proyecto `spring-security-aa-medium`)
     - Usuarios en memoria, varios roles
  3. Avanzado (proyecto `spring-security-aa-advanced`)
     - Usuarios en base de datos, varios roles

Autenticación y autorización en Spring Security

Ejemplo AA sencillo: proyecto `spring-security-aa-simple`




```
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>2.0.0.RELEASE</version>
</parent>

<dependencies>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
   </dependency>
</dependencies>
```

Autenticación y autorización en Spring Security

Ejemplo AA sencillo: proyecto `spring-security-aa-simple`

```
@Controller
public class MyController {

   @RequestMapping("/")
   public ModelAndView index() {
      return new ModelAndView("index");
   }

   @RequestMapping("/login")
   public ModelAndView login() {
      return new ModelAndView("login");
   }

   @RequestMapping("/home")
   public ModelAndView home() {
      return new ModelAndView("home");
   }

}
```

Controlador muy sencillo: sólo asocia URLs con vistas

Autenticación y autorización en Spring Security

Ejemplo AA sencillo: proyecto `spring-security-aa-simple`

```
@Configuration
@EnableGlobalMethodSecurity
public class MySecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // Paths that can be visited without authentication
        http.authorizeRequests().antMatchers("/").permitAll();
        http.authorizeRequests().antMatchers("/login").permitAll();
        http.authorizeRequests().antMatchers("/logout").permitAll();

        // Paths that cannot be visited without authentication
        http.authorizeRequests().anyRequest().authenticated();

        // Login form
        http.formLogin().loginPage("/login");
        http.formLogin().usernameParameter("username");
        http.formLogin().passwordParameter("password");
        http.formLogin().defaultSuccessUrl("/home");
        http.formLogin().failureUrl("/login?error");

        // Logout
        http.logout().logoutUrl("/logout");
        http.logout().logoutSuccessUrl("/login?logout");
    }
    // ...
```

Páginas que tendrán el acceso permitido

El resto de páginas requerirán de autenticación

Autenticación basada en formulario

Página para la desconexión

Autenticación y autorización en Spring Security

Ejemplo AA sencillo: proyecto `spring-security-aa-simple`

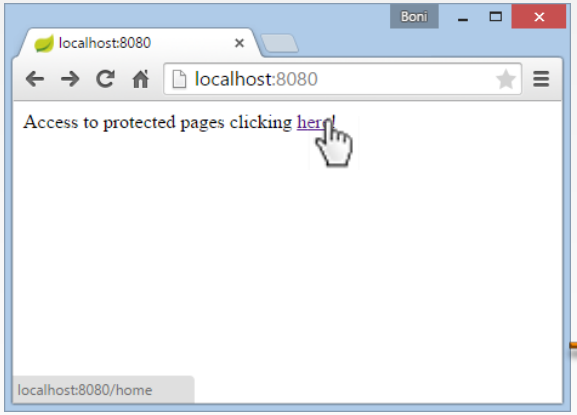
```
    // ...

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
            throws Exception {
        // Authorization
        auth.inMemoryAuthentication().withUser("user").password("{noop}p1")
                .roles("USER");
    }

}
```

Un único usuario (en memoria)

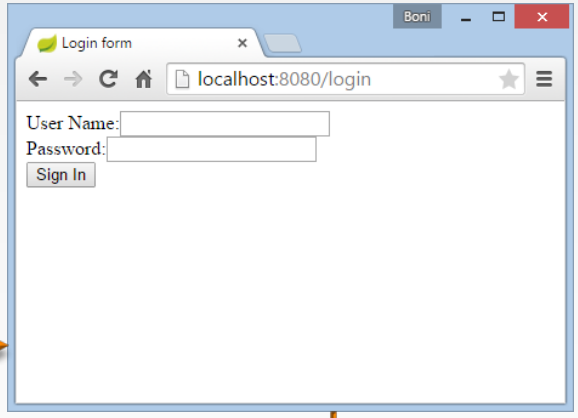
Anteponemos la etiqueta `{noop}` para guardar el password en claro (sin ningún tipo de codificación ni cifrado)

Autenticación y autorización en Spring Security

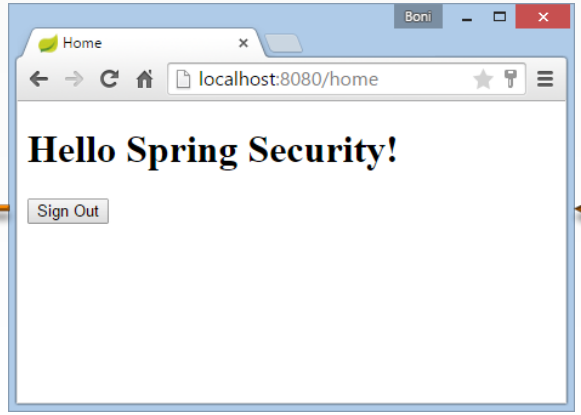
Ejemplo AA sencillo: proyecto `spring-security-aa-simple`



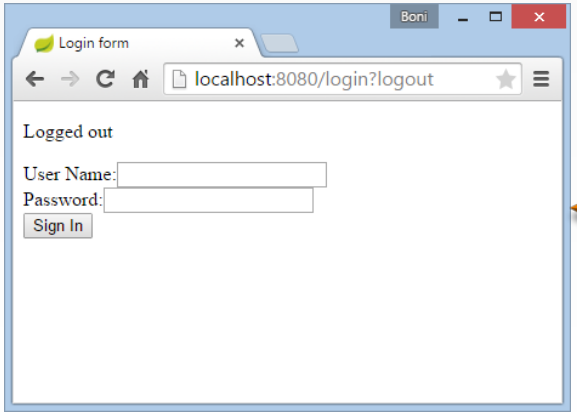
`/home` es un recurso protegido, así que se redirige la navegación al formulario de login



Con las credenciales correctas vamos a `/home`



logout



Autenticación y autorización en Spring Security

Ejemplo AA sencillo: proyecto `spring-security-aa-simple`

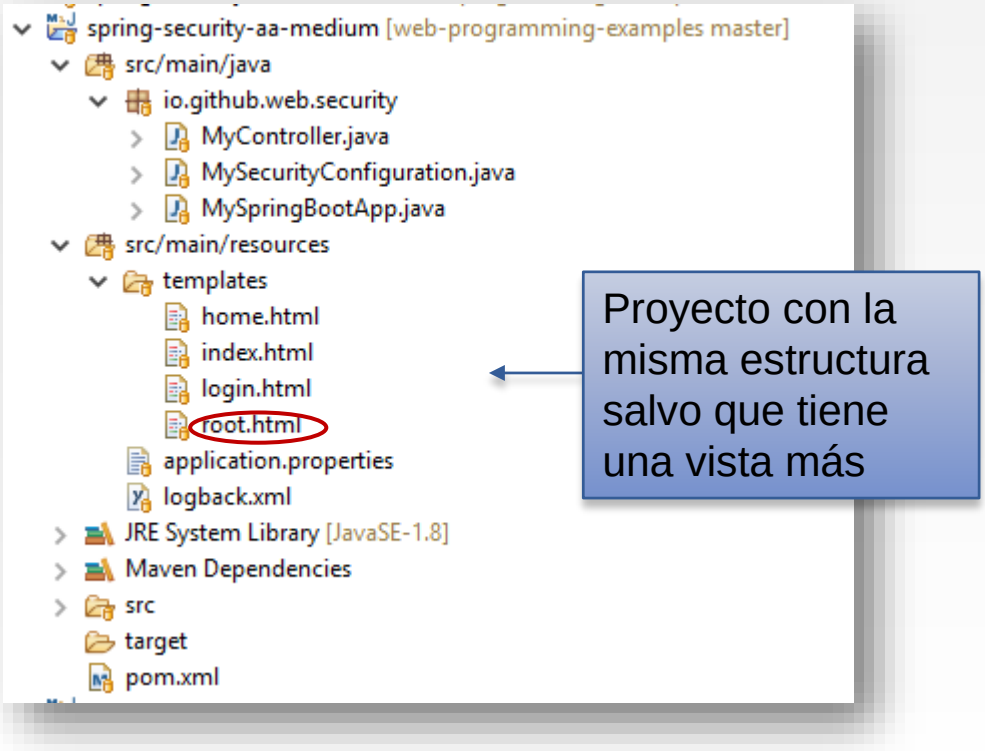
- Todas las vistas incorporan una medida de seguridad automática: un token para evitar ataques CSRF (*Cross Site Request Forgery*)
- Este token lo genera el servidor para cada petición y es requerido para poder recibir datos del cliente

```
<!DOCTYPE html>
<html>
<head>
<title>Home</title>
</head>
<body>
<h1>Hello Spring Security!</h1>
<form data-th-action="@{/logout}" method="post">
<input type="submit" value="Sign Out">
</form>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<title>Home</title>
</head>
<body>
<h1>Hello Spring Security!</h1>
<form method="post" action="/logout">
<input type="submit" value="Sign Out">
<input type="hidden" name="_csrf" value="c54a70a7-1586-
4dc3-8e64-4fac09625ce2" /></form>
</body>
</html>
```

Autenticación y autorización en Spring Security

Ejemplo AA medio: proyecto `spring-security-aa-medium`



Autenticación y autorización en Spring Security

Ejemplo AA medio: proyecto `spring-security-aa-medium`

Cambiamos la anotación que define el método de seguridad para poder restringir la autorización de los métodos controladores a ciertos roles de usuario

```
@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

   // Same authentication schema than example before

   @Override
   protected void configure(AuthenticationManagerBuilder auth)
      throws Exception {
      // Authorization
      auth.inMemoryAuthentication().withUser("user").password("{noop}p1")
         .roles("USER");
      auth.inMemoryAuthentication().withUser("root").password("{noop}p2")
         .roles("USER", "ADMIN");
   }
}
```

Dos usuarios en memoria de diferente tipo (rol)

Autenticación y autorización en Spring Security

Ejemplo AA medio: proyecto `spring-security-aa-medium`

```
@Controller
public class MyController {

    @RequestMapping("/")
    public ModelAndView index() {
        return new ModelAndView("index");
    }

    @RequestMapping("/login")
    public ModelAndView login() {
        return new ModelAndView("login");
    }

    @Secured({ "ROLE_USER", "ROLE_ADMIN" })
    @RequestMapping("/home")
    public ModelAndView home() {
        return new ModelAndView("home");
    }

    @Secured("ROLE_ADMIN")
    @RequestMapping("/root")
    public ModelAndView root() {
        return new ModelAndView("root");
    }
}
```

Los métodos protegidos se anotan con `@Secured` y el nombre del rol (con prefijo `ROLE_`)

Autenticación y autorización en Spring Security

Ejemplo AA medio: proyecto `spring-security-aa-medium`

```
<!DOCTYPE html>
<html>
<head>
<title>Home</title>
</head>
<body>
    <h1>
        Hello <span data-sec-authentication="name"></span>
    </h1>

    <div data-sec-authorize="hasRole('ROLE_ADMIN')">
        <a data-th-href="@{/root}">Administration page</a>
    </div>

    <form data-th-action="@{/logout}" method="post">
        <input type="submit" value="Sign Out">
    </form>
</body>
</html>
```

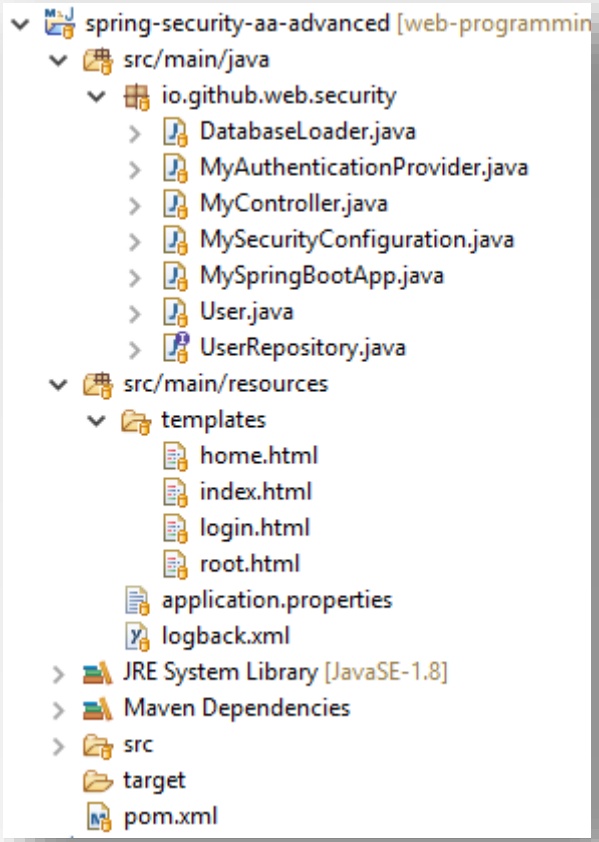
Podemos acceder a las propiedades de autenticación y autorización mediante las etiquetas Thymeleaf:
- `data-sec-authentication`
- `data-sec-authorize`

Para ello es necesario añadir la siguiente dependencia en el pom.xml

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity4</artifactId>
</dependency>
```

Autenticación y autorización en Spring Security

Ejemplo AA avanzado: proyecto `spring-security-aa-advanced`




```xml
<dependencies>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
   </dependency>
   <dependency>
       <groupId>org.thymeleaf.extras</groupId>
       <artifactId>thymeleaf-extras-springsecurity4</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
   </dependency>
   <dependency>
       <groupId>com.h2database</groupId>
       <artifactId>h2</artifactId>
   </dependency>
</dependencies>
```

Autenticación y autorización en Spring Security

Ejemplo AA avanzado: proyecto `spring-security-aa-advanced`

```
@Entity
public class User {

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   private String user;
   private String password;

   @ElementCollection(fetch = FetchType.EAGER)
   private List<GrantedAuthority> roles;

   public User() {
   }
   public User(String user, String password, List<GrantedAuthority> roles) {
      this.user = user;
      this.password = new BCryptPasswordEncoder().encode(password);
      this.roles = roles;
   }

   // getters, setters
}
```

Entidad persistente que almacena las credenciales de usuario y sus roles

Las contraseñas nunca se deben almacenar en claro (hay que cifrarlo o usar función hash)

Autenticación y autorización en Spring Security

Ejemplo AA avanzado: proyecto `spring-security-aa-advanced`

```
@Component
public class DatabaseLoader {

    private UserRepository userRepository;

    public DatabaseLoader(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @PostConstruct
    private void initDatabase() {
        // User #1: "user", with password "p1" and role "USER"
        List<GrantedAuthority> userRoles = Arrays
                .asList(new SimpleGrantedAuthority("ROLE_USER"));
        userRepository.save(new User("user", "p1", userRoles));

        // User #2: "root", with password "p2" and roles "USER" and "ADMIN"
        List<GrantedAuthority> adminRoles = Arrays.asList(
                new SimpleGrantedAuthority("ROLE_USER"),
                new SimpleGrantedAuthority("ROLE_ADMIN"));
        userRepository.save(new User("root", "p2", adminRoles));
    }

}
```

Componente usado para popular la base de datos (se ejecutará al iniciar la aplicación)

Autenticación y autorización en Spring Security

Ejemplo AA avanzado: proyecto `spring-security-aa-advanced`

```
public interface UserRepository extends CrudRepository<User, Long> {
   User findByUser(String user);
}
```

Repositorio de usuarios

```
@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class MySecurityConfiguration extends WebSecurityConfigurerAdapter {

    public MyAuthenticationProvider authenticationProvider;

    public MySecurityConfiguration(
            MyAuthenticationProvider authenticationProvider) {
        this.authenticationProvider = authenticationProvider;
    }

    // Same authentication schema than example before

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
       throws Exception {
       // Custom authentication provider
       auth.authenticationProvider(authenticationProvider);
    }
}
```

El gestor de autenticación ya no son credenciales en memoria

Autenticación y autorización en Spring Security

Ejemplo AA avanzado: proyecto `spring-security-aa-advanced`

```
@Component
public class MyAuthenticationProvider implements AuthenticationProvider {

   @Autowired
   private UserRepository userRepository;

   @Override
   public Authentication authenticate(Authentication authentication)
      throws AuthenticationException {
      String username = authentication.getName();
      String password = (String) authentication.getCredentials();
      User user = userRepository.findByUser(username);
      if (user == null) {
         throw new BadCredentialsException("User not found");
      }
      if (!new BCryptPasswordEncoder().matches(password, user.getPasswordHash())) {
         throw new BadCredentialsException("Wrong password");
      }
      List<GrantedAuthority> roles = user.getRoles();
      return new UsernamePasswordAuthenticationToken(username, password, roles);
   }
}
```

Se inyecta repositorio de usuario

Lectura de credenciales del formulario

Se comprueba usuario y contraseña

Lectura de lista de roles

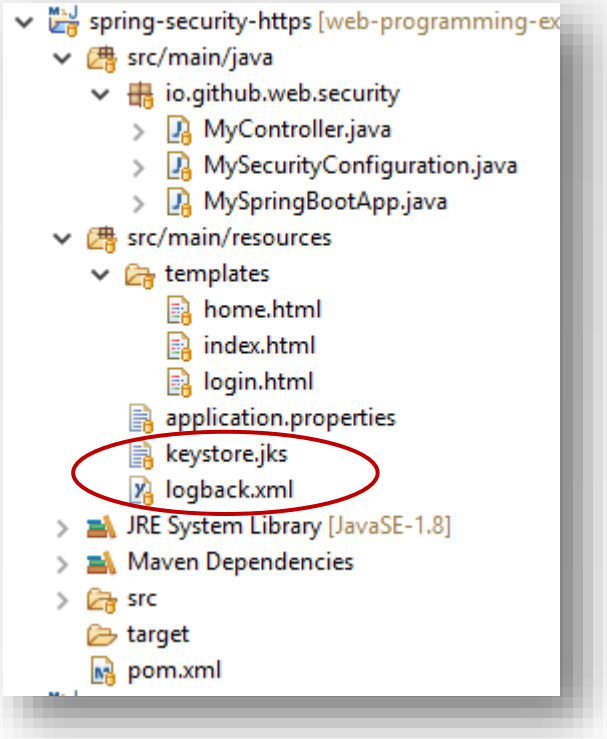
Confidencialidad en Spring Security

Ejemplo: proyecto `spring-security-https`

- Exactamente igual que proyecto `spring-security-aa-simple` excepto:
  - `application.properties`:

```
server.port = 8443
server.ssl.key-store = classpath:keystore.jks
server.ssl.key-store-password = password
server.ssl.key-password = secret
```

  - `keystore.jks`: Repositorio de certificados Java



Confidencialidad en Spring Security

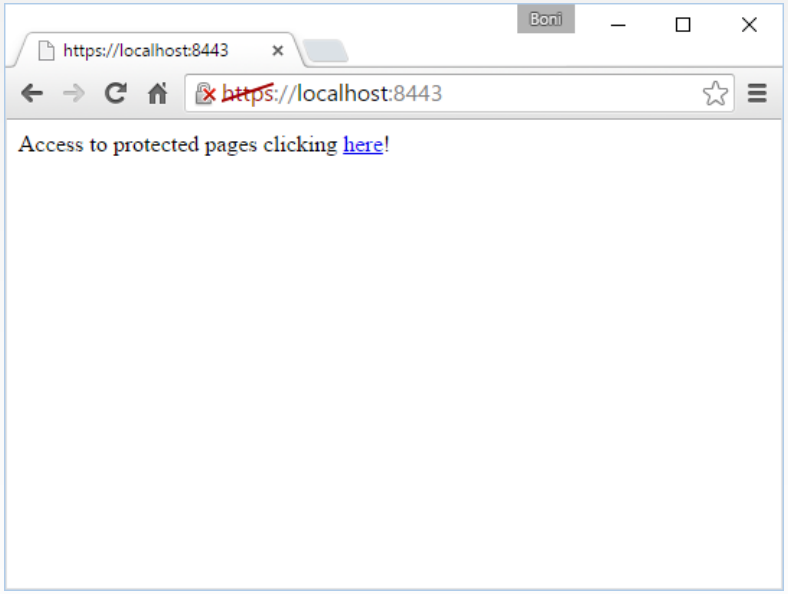
Ejemplo: proyecto `spring-security-https`

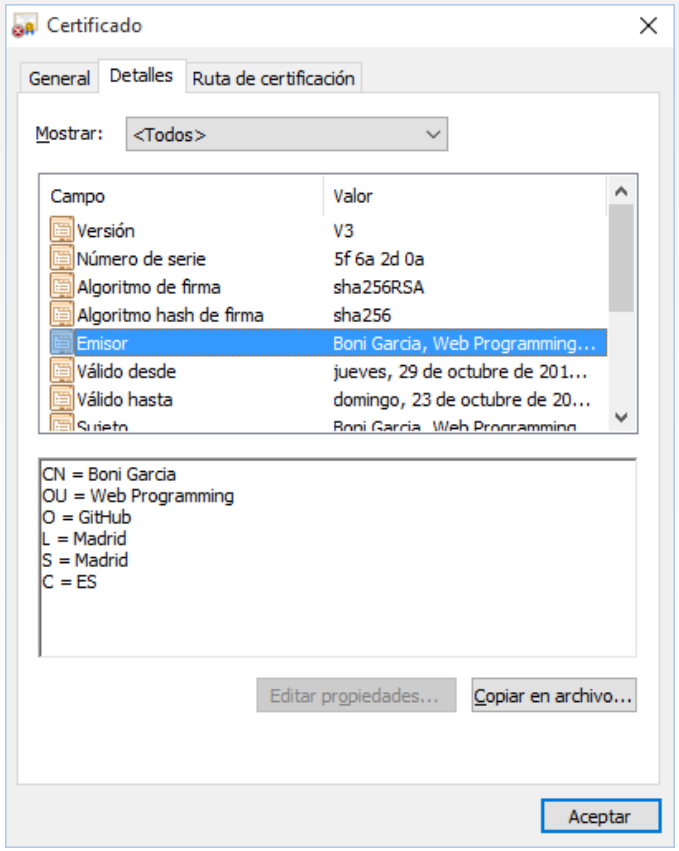
- `keystore.jks` se crea con herramienta keytool (incorporada en JRE)

```
$ cd $JAVA_HOME/bin
$ keytool -genkey -keyalg RSA -alias selfsigned -keystore keystore.jks -storepass password -
validity 360 -keysize 2048
¿Cuáles son su nombre y su apellido?
  [Unknown]:  Boni Garcia
¿Cuál es el nombre de su unidad de organización?
  [Unknown]:  Web Programming
¿Cuál es el nombre de su organización?
  [Unknown]:  GitHub
¿Cuál es el nombre de su ciudad o localidad?
  [Unknown]:  Madrid
¿Cuál es el nombre de su estado o provincia?
  [Unknown]:  Madrid
¿Cuál es el código de país de dos letras de la unidad?
  [Unknown]:  ES
¿Es correcto CN=Boni Garcia, OU=Programacion Web, O=GitHub, L=Madrid, ST=Madrid,
C=ES?
  [no]:  si
Introduzca la contraseña de clave para <selfsigned>
        (INTRO si es la misma contraseña que la del almacén de claves): secret
Volver a escribir la contraseña nueva: secret
```

Confidencialidad en Spring Security

Ejemplo: proyecto `spring-security-https`





Para ampliar: Spring Boot + Let's Encrypt
https://dzone.com/articles/spring-boot-secured-by-lets-encrypt