INGENIERÍA WEB Y COMPUTACIÓN EN LA NUBE

Bloque3: Parte servidora (backend)

TEMA 3.4: SEGURIDAD CON SPRING SECURITY

Boni García
boni.garcia@urjc.es

Universidad Rey Juan Carlos

Índice de contenidos

Índice de contenidos

1. Seguridad en redes de datos

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

1. Seguridad en redes de datos

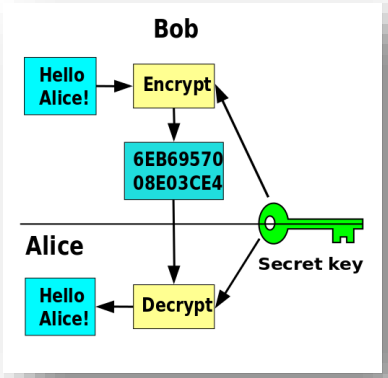
Seguridad en redes de datos

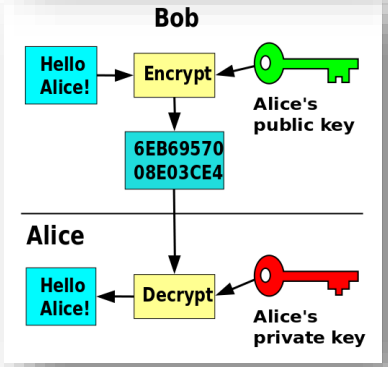
- La **autenticación** se consigue mediante:
  - Algo que sabes. Por ejemplo, unas credenciales login-password
  - Algo que tienes. Por ejemplo, una tarjeta de acceso
  - Algo que eres. Por ejemplo, cualidades biométricas (huella digital…)
- La **autorización** discrimina el acceso a un determinado recurso en base a permisos (*grants*), lista de control de acceso (*Access Control List*, ACL), políticas (*policies*), roles, tokens, …
  - Normalmente 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

1. Seguridad en redes de datos

Seguridad en redes de datos

- La **confidencialidad** se consigue típicamente 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**





1. Seguridad en redes de datos

Servicios de seguridad

- En los criptosistemas de clave pública, un **certificado digital** es un documento electrónico que asocia el nombre de una entidad con su clave pública durante un determinado periodo de validez
- El certificado digital es emitido por una Autoridad de Certificación (CA) , o sea, la "Tercera Parte de Confianza" (TTP, *Trusted Third Party*)
- Los certificados usados en aplicaciones web asocian un dominio web a su clave pública
- Tipo de certificado más utilizado: X.509 (estándar del ITU-T)

1. Seguridad en redes de datos

Servicios de seguridad

- Ejemplos de algoritmos:

Criptosistemas asimétricos

- RSA (Rivest, Shamir y Adleman)
- Diffie-Hellman
- ElGamal
- Criptografía de curva elíptica

Funciones hash

- SHA (*Secure Hash Algorithm*)
- MD5 (*Message-Digest Algorithm 5*)
- DSA (*Digital Signature Algorithm*)

Criptosistemas simétricos

- AES (*Advanced Encryption Standard*)
- ES (*Data Encryption Standard*)
- IDEA (*International Data Encryption Algorithm*)
- 3DES
- RC2, RC4, RC5
- Blowfish

1. Seguridad en redes de datos

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 SSL/TLS son:
  - Confidencialidad: se cifra el intercambio de datos a nivel TCP
  - Autenticación: entidades pueden confirmar su identidad. En un protocolo cliente-servidor sólo el servidor es autenticado (es decir, se garantiza su identidad) mientras que el cliente se mantiene sin autenticar
  - Integridad: se usa una función hash para garantizar la integridad de datos

1. Seguridad en redes de datos

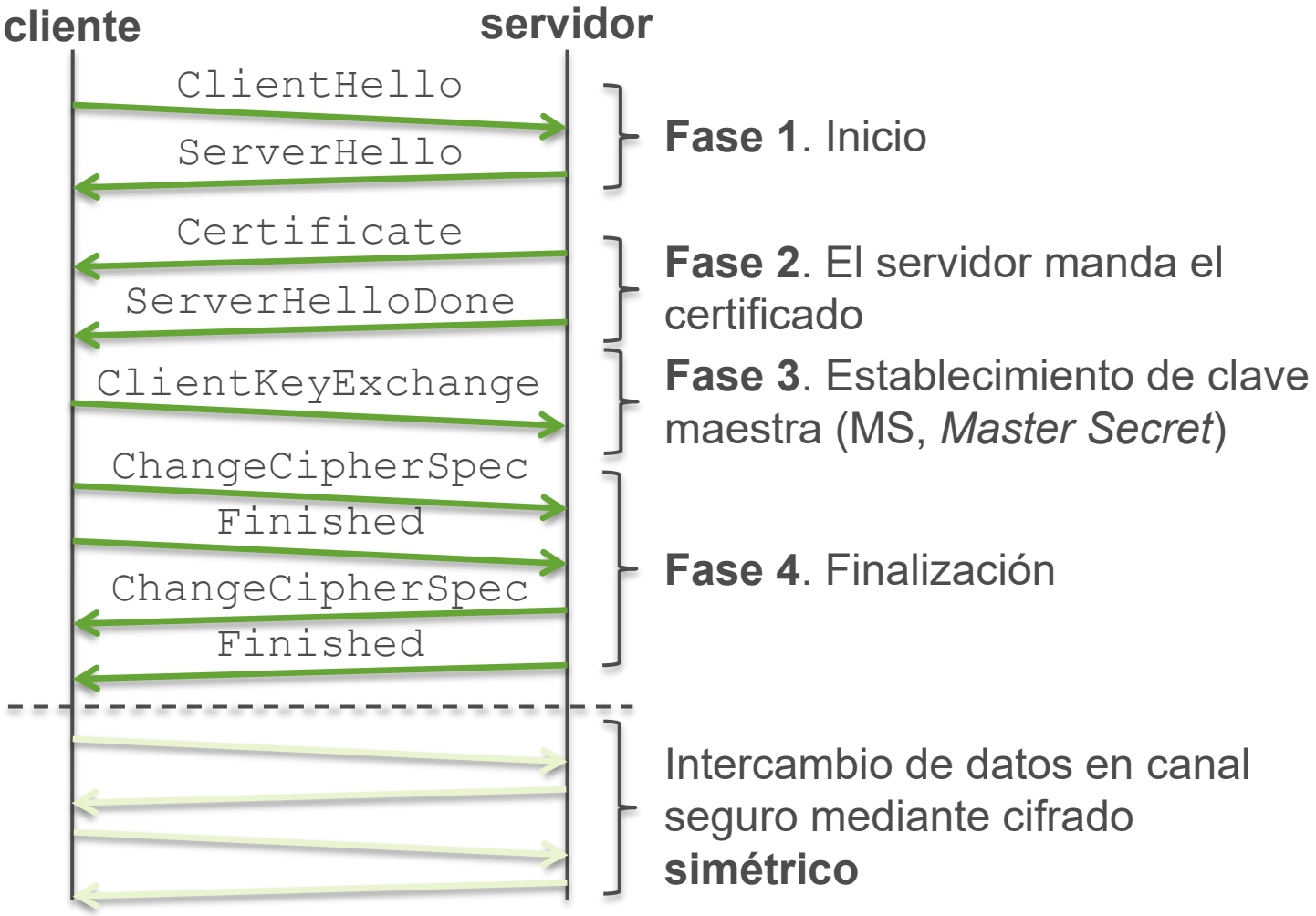
TLS

- Para establecer un canal seguro cifrado, las entidades tienen que llegar a un acuerdo (***handshake***)
- El *handshake* tiene varias fases:
  1. Establecimiento de características seguridad (algoritmo cifrado, etc)
  2. Servidor envía su **certificado** digital
  3. Cliente establece la clave maestra (MS, *Master Secret*)
     - Opcionalmente el cliente puede mandar su certificado en esta fase
     - En aplicaciones web (HTTPS), el servidor es el único autenticado, es decir, sólo el servidor envía un certificado al cliente
  4. Finalización del *handshake* e inicio de comunicación segura
- Después del *handshake* el intercambio de tráfico se produce mediante cifrado del tráfico basado en cifrado **simétrico**

1. Seguridad en redes de datos

TLS

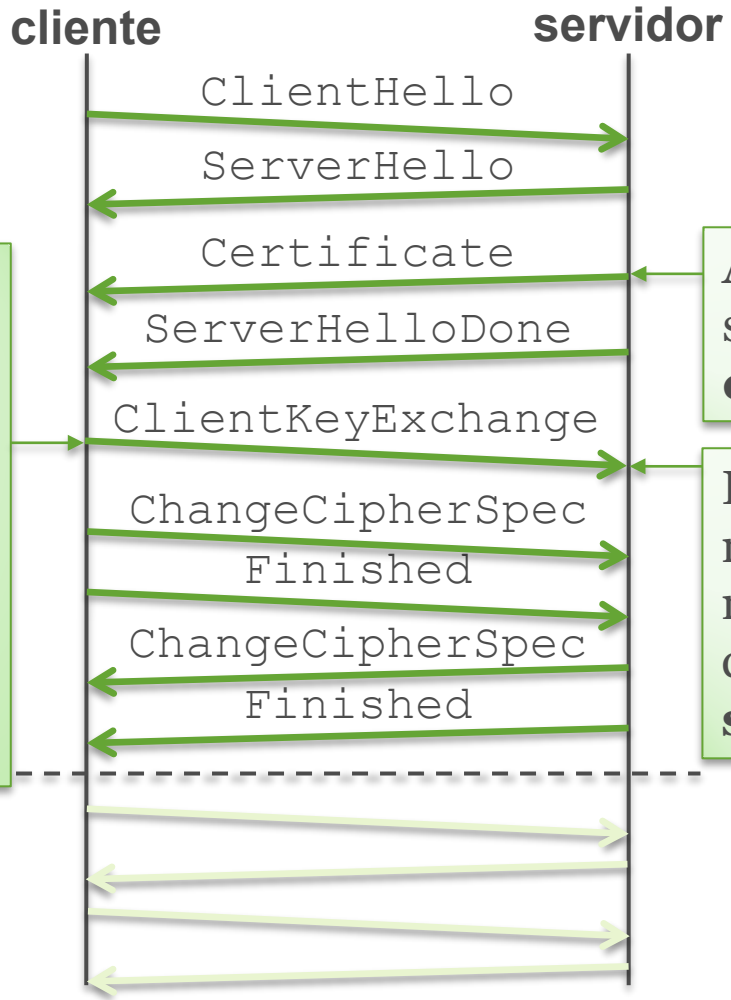
- Handshake:



1. Seguridad en redes de datos

TLS

- Handshake:



El cliente genera una clave maestra MS que será usada para cifrar todos los datos de la sesión segura. Esta clave se envía cifrada con la **clave pública del servidor**, obtenida a partir del certificado

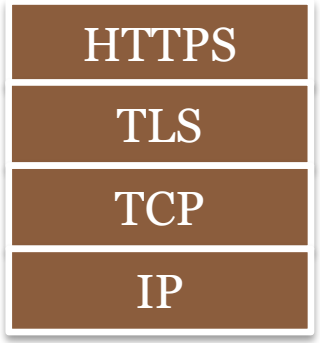
Al enviar el certificado el servidor está mandando su **clave pública** al cliente

El servidor obtiene la clave maestra MS descifrando el mensaje enviado por el cliente, usando para ello **su clave privada**

1. Seguridad en redes de datos

HTTPS

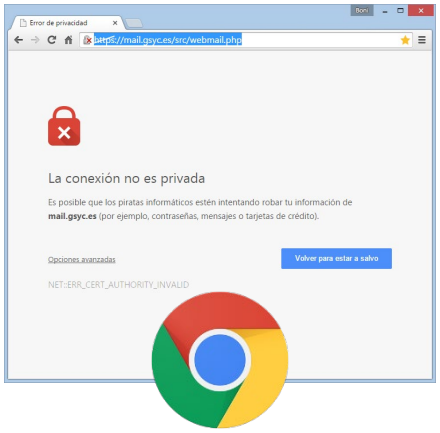
- *Hypertext Transfer Protocol Secure*. Versión segura de HTTP
- HTTPS no es más que HTTP sobre TLS/SSL
- Con HTTPS se consigue que la información sensible (claves, etc) no pueda ser interceptada por un atacante, ya que lo único que obtendrá será un flujo de datos cifrados que le resultará imposible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

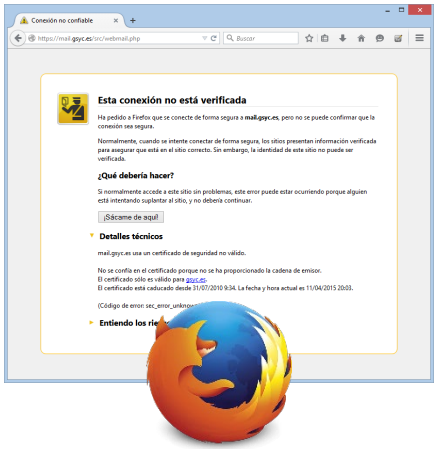


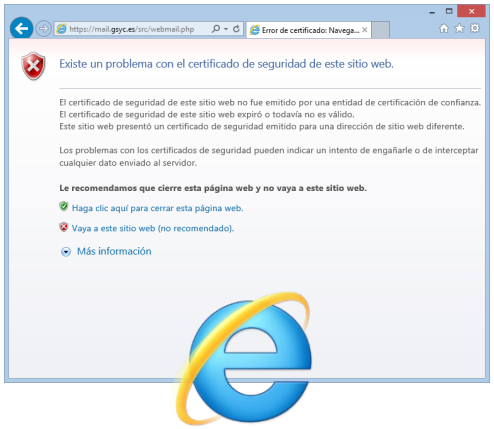
1. Seguridad en redes de datos

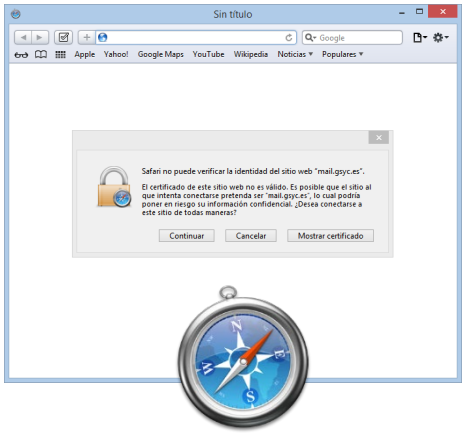
HTTPS

- Los navegadores tienen una lista de CAs conocidas
- Al recibir un certificado no valido muestra una alerta de seguridad al usuario. Esto ocurre cuando:
  - El certificado firmado por una CA no conocida (por ejemplo, un certificado autofirmado)
  - El certificado ha caducado





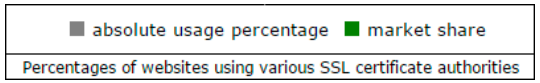


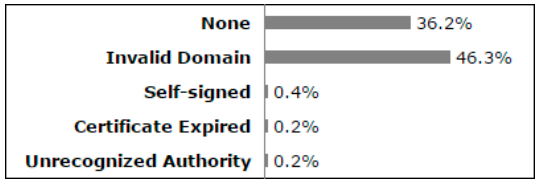


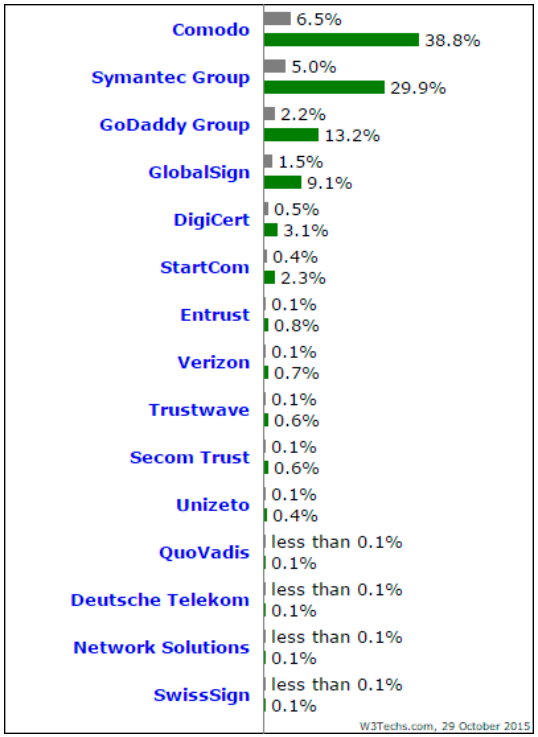
1. Seguridad en redes de datos

HTTPS

- Estadísticas de uso de CAs en aplicaciones web (abril de 2015)
- Fuente: http://w3techs.com/technologies/overview/ssl_certificate/all







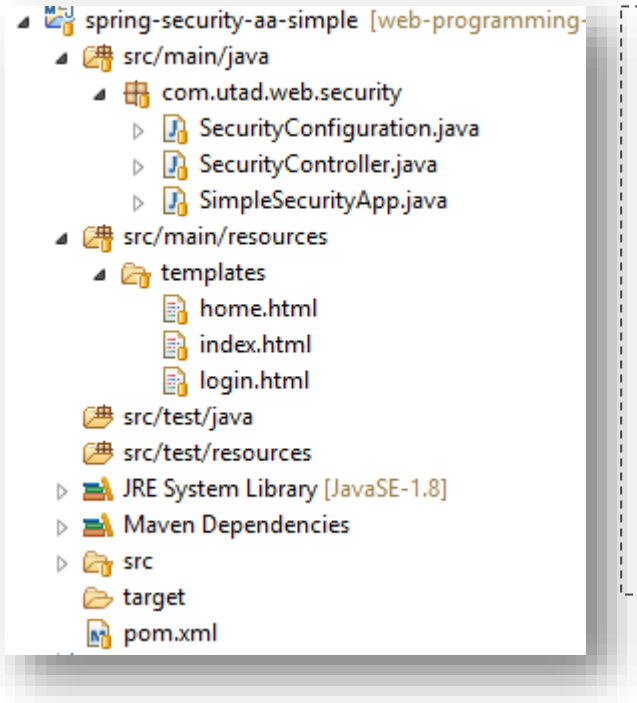
Índice de contenidos

2. Autenticación y autorización

- Vamos a ver como implementar los servicios de seguridad de autenticación y autorización en Spring (Boot + 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

2. Autenticación y autorización

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




```xml
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>1.2.7.RELEASE</version>
</parent>

<dependencies>
   <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>
```

2. Autenticación y autorización

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

```
@Controller
public class SecurityController {

    @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

2. Autenticación y autorización

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

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

   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http.authorizeRequests().antMatchers("/").permitAll().anyRequest()
        .authenticated();

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

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

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

El path "/" tendrá acceso permitido. El resto necesitará autenticación

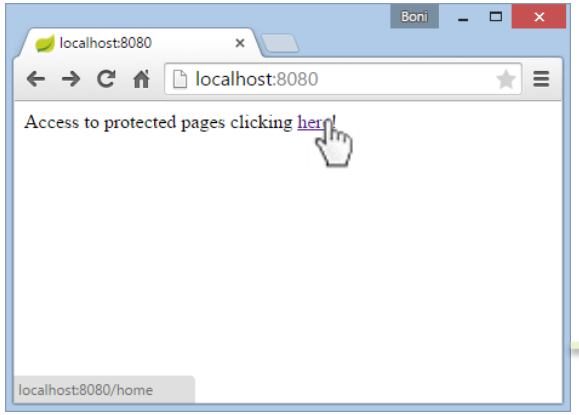
Autenticación basada en formulario

Página para la desconexión

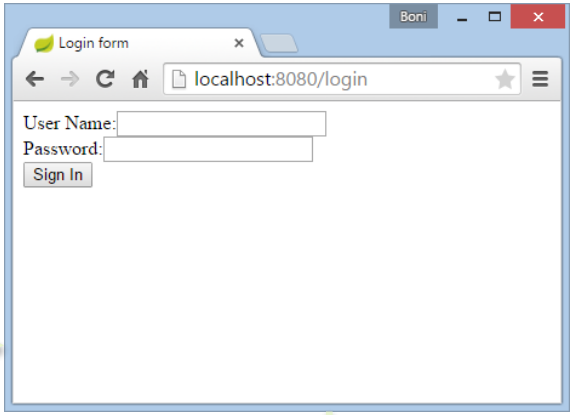
Un único usuario (en memoria)

2. Autenticación y autorización

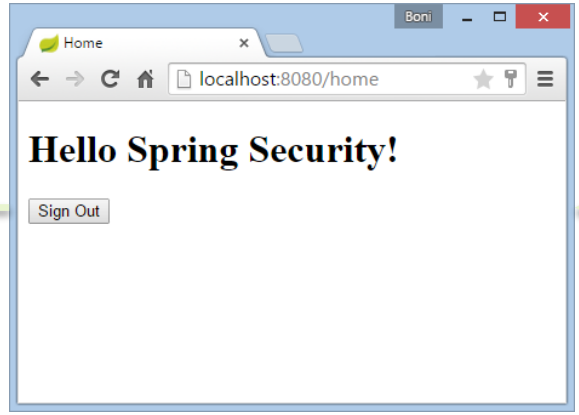
Ejemplo AA sencillo: proyecto `spring-security-aa-simple`



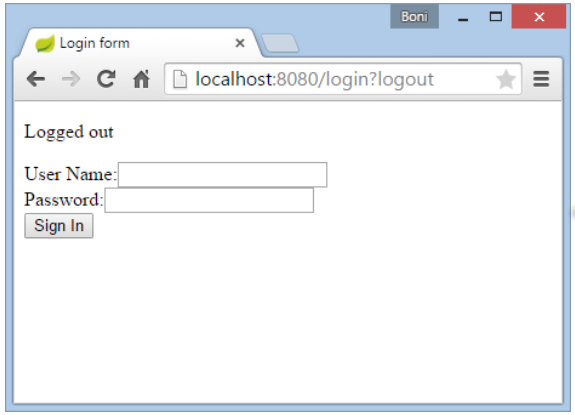
`/home` es un recurso protegido, así que se redirige la navegación al formulario de login



Con las credenciales correctas vamos a `/home`



logout



2. Autenticación y autorización

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

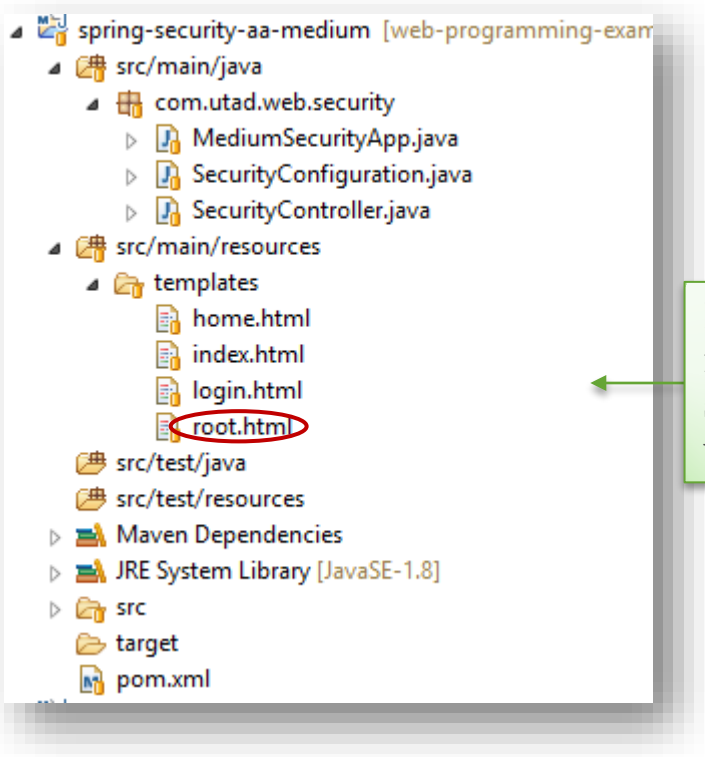
- Todas las vistas incorporan una medida de seguridad automática: un token automático CSRF (*Cross Site Request Forgery*)
- Este token lo genera el servidor para cada petición y es requerido para poder recibir datos del cliente

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>Home</title>
</head>
<body>
<h1>Hello Spring Security!</h1>
<form 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>
```

2. Autenticación y autorización

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



Proyecto con la misma estructura salvo que tiene una vista más

2. Autenticación y autorización

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("p1")
         .roles("USER");
      auth.inMemoryAuthentication().withUser("root").password("p2")
         .roles("USER", "ADMIN");
   }
}
```

Dos usuarios en memoria de diferente tipo (rol)

2. Autenticación y autorización

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

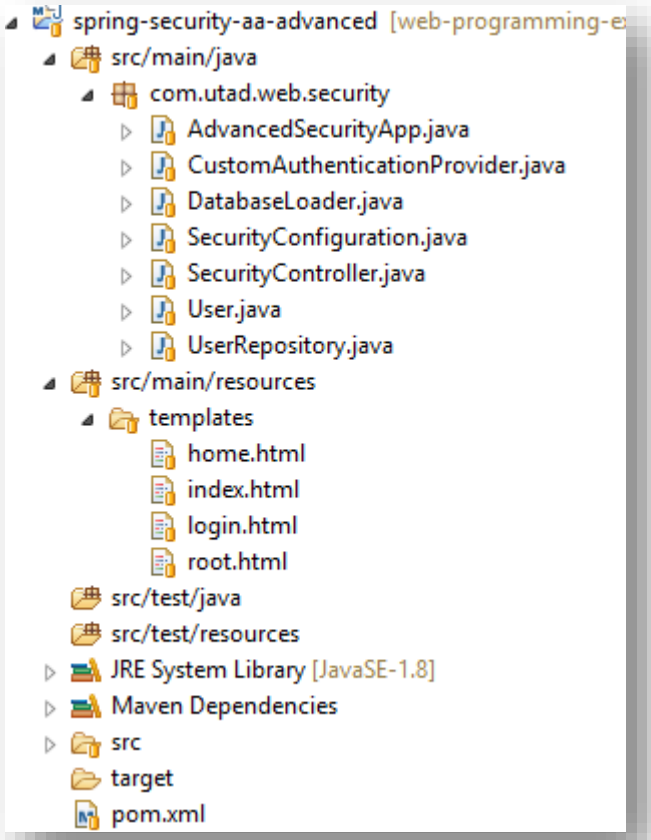
```java
@Controller
public class SecurityController {

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

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

```java
    @Secured({ "ROLE_USER", "ROLE_ADMIN" })
    @RequestMapping("/home")
    public ModelAndView home() {
        Authentication auth = SecurityContextHolder
            .getContext().getAuthentication();
        String name = auth.getName();
        ModelAndView model = new ModelAndView("home")
            .addObject("name", name);

        if (auth.getAuthorities().contains(
            new SimpleGrantedAuthority("ROLE_ADMIN"))) {
            model = model.addObject("admin", true);
        }
        return model;
    }

    @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_`)

2. Autenticación y autorización

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




```
<dependencies>
   <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.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
   </dependency>

   <dependency>
       <groupId>com.h2database</groupId>
       <artifactId>h2</artifactId>
   </dependency>
</dependencies>
```

2. Autenticación y autorización

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

```
@Component
public class DatabaseLoader {

    @Autowired
    private UserRepository userRepository;

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

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

}
```

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

2. Autenticación y autorización

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 almacenará las credenciales de usuario y sus roles

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

2. Autenticación y autorización

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 SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Autowired
    public CustomAuthenticationProvider authenticationProvider;

    // Same authentication schema than examples before

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

El gestor de autenticación ya no son credenciales en memoria

2. Autenticación y autorización

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

```
@Component
public class CustomAuthenticationProvider 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

Índice de contenidos

1. Seguridad en redes de datos
2. Autenticación y autorización
3. Confidencialidad
   - Ejemplo AA sencillo
   - Ejemplo AA medio
   - Ejemplo AA avanzado

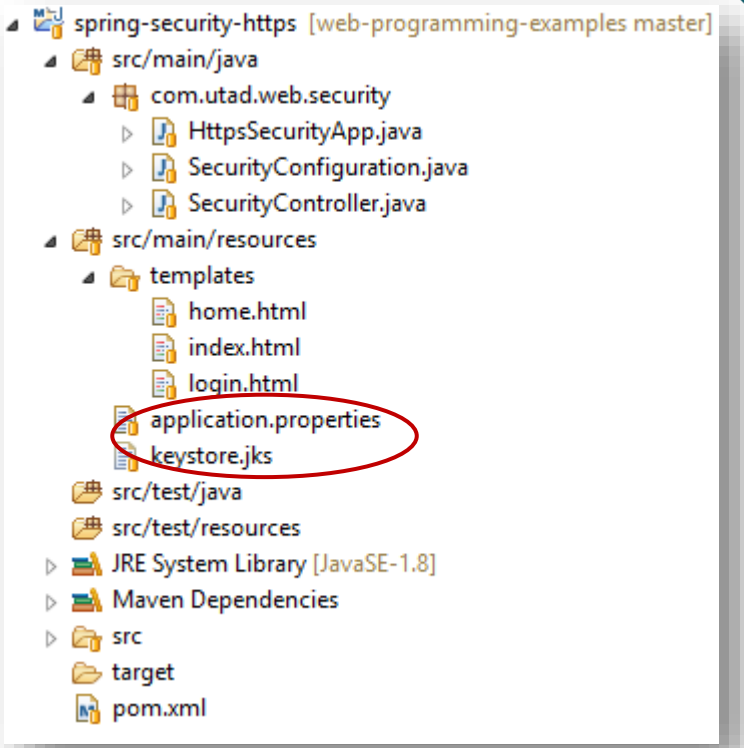
3. Confidencialidad

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



3. Confidencialidad

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=U-tad, 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
```

3. Confidencialidad

Ejemplo: proyecto `spring-security-https`