



Componentes Web

Sesión 6 - Seguridad en aplicaciones web



Índice

- Seguridad en aplicaciones web
- Realms de autenticación
- Autenticación basic
- Autenticación mediante formularios
- Anotaciones de seguridad



Seguridad de aplicaciones web

- *Autenticación y autorización*
 - Autenticación: ¿Quién accede al recurso?
 - Autorización: ¿Qué operaciones puede hacer?
 - Credenciales (usuario y password / certificados digitales)
- *Confidencialidad*
 - Asegurar que sólo los elementos que intervienen entienden el proceso de comunicación
 - Transporte cifrado mediante SSL
- *Integridad*
 - Verificar que el contenido de la información no se ve modificado durante la transmisión
 - Firma digital



Seguridad declarativa y programada

- Seguridad declarativa
 - Se declara en el descriptor de despliegue
 - La gestiona el propio servidor de aplicaciones
 - No es necesario introducir código en los componentes web
 - Puede requerir la definición de algunas páginas html
- Seguridad programada
 - Debemos implementar la seguridad en nuestros componentes web
 - Nuestro código es responsable de la gestión de la seguridad
 - Debemos verificar permisos en el código
 - Más flexible pero más tedioso
 - Cambios en seguridad normalmente implican cambios de código



Realms

- Conjunto de usuarios + passwords + roles
 - Los usuarios y passwords identifican a los clientes (*autenticación*)
 - Los roles determinan permisos en una aplicación web (*autorización*)
- Se utilizan para implementar seguridad declarativa
- *Realms* por defecto en WildFly
 - **ManagementRealm**: Usuarios para la administración del servidor
 - **ApplicationRealm**: Usuarios y roles para las aplicaciones instaladas en el servidor
 - Ambos se implementan en ficheros de tipo `.properties`
 - Podemos dar de alta usuarios con el comando `$WILDFLY_HOME/bin/addUser.sh`

Podemos definir otros *realms*, por ejemplo para obtener los usuarios de una base de datos o mediante LDAP



Tipos de autenticaciones

- **Basic:** proporcionado por HTTP. Basado en cabeceras de autenticación. Codificación Base64.
- **Digest:** similar, pero con codificación MD5
- **Basada en formularios:** el usuario configura el formulario de login en una página HTML
- **Certificados y SSL:** basado en criptografía de clave pública, es una capa entre TCP/IP y HTTP que garantiza la confidencialidad e integridad.



Autenticación declarativa

- Asociar un *realm* a la aplicación o utilizar el que herede del contexto superior
- Establecer logins, passwords y roles
- Indicar qué recursos se quiere proteger, y a qué roles serán accesibles
- Según el mecanismo de control
 - Formularios: definir página HTML con formulario de login y página de "No autorizado"
 - BASIC: nada. El navegador saca el cuadro de diálogo



Autenticación *basic*

1. Establecer logins, password y roles

- Igual que para la autenticación basada en formularios



Autenticación *basic*

2. Indicar autenticación basic

- Añadimos grupo *login-config* en fichero *web.xml*

```
<login-config>  
  <auth-method>BASIC</auth-method>  
  <realm-name>dominio</realm-name>  
</login-config>
```



Autenticación *basic*

3. Indicar qué direcciones proteger

- Añadir bloque *security-constraint* en *web.xml*

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>
      Prueba
    </web-resource-name>
    <url-pattern>
      /prueba/*
    </url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>admin</role-name>
    <role-name>subadmin</role-name>
  </auth-constraint>
</security-constraint>
```



Ventajas y problemas

- Ventajas
 - Simple de definir
 - No necesita *cookies*
- Problemas
 - Cuadro de diálogo de *login* no configurable
 - Para entrar como otro usuario hay que cerrar el navegador



Autenticación basada en formularios

1. Establecer logins, password y roles

- Hemos de definir un realm o utilizar uno existente



Autenticación basada en formularios

2. Indicar autenticación basada en formularios en el web.xml (etiqueta login-config)

- Añadimos *login-config* en web.xml

```
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>
      /login.jsp
    </form-login-page>
    <form-error-page>
      /error.html
    </form-error-page>
  </form-login-config>
</login-config>
```



Autenticación basada en formularios

3. Crear página de login

- Acción: j_security_check
- Método: POST
- Campo login: j_username
- Campo password: j_password

```
<form action="j_security_check" METHOD="POST">  
  Login:<input type="text" name="j_username"/><br>  
  Password:<input type="text" name="j_password"/><br>  
  <input type="submit" value="Enviar"/>  
</form>
```



Autenticación basada en formularios

4. Crear página de error

- Cualquier página HTML o JSP con el adecuado formato y mensaje de error

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<body>
  <h1>ERROR AL AUTENTIFICAR USUARIO</h1>
</body>
</html>
```



Autenticación basada en formularios

5. Indicar qué direcciones proteger

- Añadir bloque *security-constraint* en *web.xml*

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>
      Prueba
    </web-resource-name>
    <url-pattern>
      /prueba/*
    </url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>admin</role-name>
    <role-name>subadmin</role-name>
  </auth-constraint>
</security-constraint>
```




Ventajas y problemas

- Ventajas
 - “Look” del formulario de *login* totalmente personalizable
 - Es fácil salir y entrar como otro usuario
- Problema
 - Confía en el uso de *cookies*



Anotaciones de seguridad

- A partir del API de servlets 3.0
- Son anotaciones exclusivas para servlets

```
@WebServlet("/MiServlet")
@ServletSecurity(@HttpConstraint(rolesAllowed={"rol1","rol2"}))
public class MiServlet extends HttpServlet { ... }
```

- Configuración para cada comando HTTP

```
@ServletSecurity(
    value=@HttpConstraint(EmptyRoleSemantic.PERMIT),
    httpMethodConstraints={
        @HttpMethodConstraint(value="POST",rolesAllowed="admin")
        @HttpMethodConstraint(value="PUT",
            transportGuarantee=TransportGuarantee.CONFIDENTIAL)})
public class MiServlet extends HttpServlet { ... }
```



Acceso a la información de seguridad

- Principal del usuario

```
Principal p = request.getUserPrincipal();
if(p!=null) {
    out.println("El usuario autenticado es " + p.getName());
} else {
    out.println("No hay ningun usuario autenticado");
}
```

- Comprobación de roles

```
if(request.isUserInRole("admin")) {
    usuarioDao.altaUsuario(usuario);
} else {
    out.println("Solo los administradores pueden realizar altas");
}
```



¿Preguntas?