



Servidores Web

- Sesión 3: Seguridad en aplicaciones web



Puntos a tratar

- Seguridad del servidor
 - Políticas de seguridad
 - Autenticación en Tomcat: Realms
- Seguridad en aplicaciones web
 - Tipologías
 - Autenticación basada en formularios
 - Autenticación *basic*



Políticas de seguridad

- Evitan que las aplicaciones accedan a recursos no permitidos o hagan operaciones no autorizadas
- Fichero **catalina.policy**: fichero estándar de políticas de seguridad
 - Clases de Tomcat + JDK: todos los permisos
 - Aplic. Web: leer props del sistema, JNDI, etc.
- Por defecto, Tomcat se ejecuta sin políticas de seguridad, para utilizarlas hacer

```
startup -security
```



Realms

- Conjunto de usuarios + passwords + roles
- Los roles determinan permisos en una aplicación web
- Implementaciones básicas para realms
 - `UserDatabaseRealm`: empleada por defecto en `server.xml`. Almacena los usuarios en un fichero xml (*tomcat-users.xml*)
 - `JDBCRealm`: almacena los usuarios en una base de datos accesible mediante JDBC



JDBCRealm

Atributo	Significado
className	clase Java que implementa este <i>realm</i> . Debe ser <code>org.apache.catalina.realm.JDBCRealm</code>
connectionName	nombre de usuario para la conexión JDBC
connectionPassword	password para la conexión JDBC
connectionURL	URL de la base de datos
debug	nivel de depuración. Por defecto 0 (ninguno). Valores más altos indican más detalle.
digest	Algoritmo de "digest" (puede ser SHA, MD2 o MD5). Por defecto es <code>cleartext</code>
driverName	clase Java que implementa el <i>driver</i> de la B.D.
roleNameCol	nombre del campo que almacena los roles
userNameCol	nombre del campo que almacena los <i>logins</i>
userCredCol	nombre del campo que almacena los <i>passwords</i>
userRoleTable	nombre de la tabla que almacena la relación entre <i>login</i> y roles
userTable	nombre de la tabla que almacena la relación entre <i>login</i> y <i>password</i>



Seguridad de aplicaciones web

- Motivaciones
 - Prevenir acceso a recursos no autorizados dentro de una aplicación
 - Encriptar datos en transporte
- Aspectos:
 - *Autenticación*: saber quién accede al recurso
 - *Confidencialidad*: asegurar que sólo los elementos que intervienen entienden el proceso de comunicación
 - *Integridad*: verificar que el contenido de la información no se ve modificado durante la transmisión



Control de la autenticación

- Seguridad declarativa
 - Se configura en el web.xml
 - Puede requerir la definición de algunas páginas html
 - Ningún cambio de código
- Seguridad programada
 - El programador es responsable de verificar permisos
 - Más flexible pero más tedioso
 - Cambios en seguridad normalmente implican cambios de código



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 basada en formularios (1/5)

1. Establecer logins, password y roles

- Hemos de definir un realm:
- Podemos usar el *UserDatabaseRealm* que viene por defecto, y añadir los usuarios que queramos al fichero
\${tomcat.home}/conf/tomcat-users.xml
- Podemos definir un *JDBCRealm* y crear una base de datos con usuarios, y asociarla luego al contexto de nuestra aplicación (etiqueta *Realm* dentro del *Context* de la aplicación), o a nivel global para el servidor



Autenticación basada en formularios (2/5)

2. Indicar autenticación basada en formularios

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

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



Autenticación basada en formularios (3/5)

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 (4/5)

4. Crear página de error

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

```
<html>
<body>
    <h1>Error en la aplicación</h1>
</body>
</html>
```



Autenticación basada en formularios (5/5)

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>
<login-config>
```

...



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*



Autenticación *basic* (1/3)

1. Establecer logins, password y roles

- Igual que para la autenticación basada en formularios



Autenticación *basic* (2/3)

2. Indicar autenticación basic

- Añadimos grupo *login-config* en fichero *web.xml* pero diferente a la autenticación basada en formularios

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



Autenticación *basic* (3/3)

3. Indicar qué direcciones proteger

- Añadir bloque *security-constraint* en *web.xml*, igual que para la autenticación basada en formularios



Ventajas y problemas

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