Servidores Web

Sesión 5: Seguridad y autentificación en Tomcat

Puntos a tratar

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

Políticas de seguridad

- Evitan que las aplicaciones accedan a recursos no permitidos o hagan operaciones no autorizadas

- Fichero **conf/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
```

Seguridad de aplicaciones web

- Motivaciones
  - Prevenir acceso a recursos no autorizados dentro de una aplicación
  - Encriptar datos en transporte

- Aspectos:
  - *Autentificación*: saber quién accede al recurso (login/password)
  - *Autorización:* saber que quien accede tiene permiso para hacerlo (rol)
  - *Confidencialidad*: asegurar que sólo los elementos que intervienen entienden el proceso de comunicación (SSL)
  - *Integridad*: verificar que el contenido de la información no se ve modificado durante la transmisión (SSL)

Realms en Tomcat

- 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 *realm*. Debe ser `org.apache.catalina.realm.JDBCRealm`
`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 `cleartext`
`driverName`	clase Java que implementa el *driver* de la B.D.
`roleNameCol`	nombre del campo que almacena los roles
`userNameCol`	nombre del campo que almacena los *logins*
`userCredCol`	nombre del campo que almacena los *passwords*
`userRoleTable`	nombre de la tabla que almacena la relación entre *login* y roles
`userTable`	nombre de la tabla que almacena la relación entre *login* y *password*

JDBCRealm (II)

- Se puede configurar en **META-INF/context.xml** si es solo para una aplicación
  - Si se va a usar en varias, se podría poner en el **server.xml**

```
<Context>
<Realm
      className="org.apache.catalina.realm.JDBCRealm"
      connectionName="root"
      connectionPassword="root"
      connectionURL="jdbc:mysql://localhost:3306/ejemplo"
      driverName="com.mysql.jdbc.Driver"
      roleNameCol="rol"
      userNameCol="login"
      userCredCol="password"
      userRoleTable="roles"
      userTable="usuarios"  />
</Context>
```

Control de la autentificación

- 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

Tipos de autentificaciones

- Basic: proporcionado por HTTP. Basado en cabeceras de autentificación. 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.

Autentificación 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

Autentificación 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

Autentificación basada en formularios (2/5)

2. Indicar autentificación 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>
```

Autentificación 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>
```

Autentificación 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>
```

Autentificación basada en formularios (5/5)

5. Indicar qué direcciones proteger

- Añadir bloque *security-constraint* en *web.xml.* Por cada rol, añadir un *security-role*

```
<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>
<security-role>
  <description>administrador de la web</description>
  <role-name>admin</role-name>
</security-role>
```

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*

Autentificación *basic* (1/3)

1. Establecer logins, password y roles

- Igual que para la autentificación basada en formularios

Autentificación *basic* (2/3)

2. Indicar autentificación basic

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

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

Autentificación *basic* (3/3)

3. Indicar qué direcciones proteger

- Añadir bloque *security-constraint* en *web.xml*, igual que para la autentificación 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