

## Seguridad de la aplicación para servlets y JSP (página activas java)

- Autenticación de Servlets (java)
  - Usuarios y roles.
  - Autenticación declarativa.
  - Portabilidad
  - Tipos de autenticación
    - Básica.
    - Digest.
    - Basada en el formulario.
    - SSL y certificados.
- Elementos de seguridad de aplicaciones web.
- Autenticación controlada. (usada por el sistema)

### Autenticación con servlets

Básicamente puede considerarse este esquema:

1. Un usuario intenta obtener acceso a un recurso protegido como una página JSP.

2. Si el usuario es autenticado, el servlet brinda acceso a la página, si no pregunta por usuario y password.

3. Si el usuario y password no puede ser autenticado, se presenta un mensaje y se da la oportunidad de ingresar nuevamente el usuario y password.

Este esquema puede completarse con: quién pregunta por el usuario y password, quién es el responsable de la autenticación, cómo se realiza la autenticación o cuantas veces puede el usuario intentar ingresar.

### Usuarios y roles

Un usuario puede cumplir uno o varios roles, por ejemplo el Jefe de Estudios puede ser también un Cliente (un padre de familia).

Cuando se trabaja con un servidor de aplicaciones Web como el Jakarta-Tomcat (Apache) las restricciones de seguridad que se encuentra en el archivo WEB-INF/WEB.XML asocian roles y recursos protegidos como páginas jsp. Como se muestra en el siguiente ejemplo:

---

```

<web-app>
...
<security-constraint>
  <!--recursos web que se protegerán-->
  <web-resource-collection>
    <web-resource-name>RECURSO PROTEGIDO</web-resource-name>
    <url-pattern>/Criterios.jsp</url-pattern>
  </web-resource-collection>

  <auth-constraint>
    <!--Roles que tendrán acceso a los recursos especificados arriba-->
    <role-name>Diseñador-esquemas</role-name>
  </auth-constraint>
</security-constraint>
...
<security-constraint>
  <!--recursos web que se protegerán-->
  <web-resource-collection>
    <web-resource-name>RECURSO PROTEGIDO</web-resource-name>
    <url-pattern>/Cursos.jsp</url-pattern>
  </web-resource-collection>

  <auth-constraint>
    <!--Roles que tendrán acceso a los recursos especificados arriba-->
    <role-name>Docente</role-name>
  </auth-constraint>
</security-constraint>
</web-app>

```

En este ejemplo se muestra dos restricciones de acceso vía WEB.XML para los roles Docente y Diseñador de Esquemas. Se puede apreciar la asociación entre rol y recurso protegido. Las restricciones de seguridad como esta son manejadas con un contenedor de servlets o aplicación como Tomcat en la cual se edita el archivo tomcat-users.xml

```

<tomcat-users> ... <user name="rgomez" password="plomo" roles="docente", "cliente"/> ...
</tomcat-users>

```

Luego la clase `HttpServletRequest` tendrá métodos como: “boolean `isUserInRole(Usuario)`” y “String `getRemoteUser()`” que accederán estos datos de seguridad.

Este mecanismo de seguridad que puede implementarse es manejado sólo por los contenedores de servlets, es decir, las aplicaciones web, no lo pueden setear. Esto implica que se debe escribir en el archivo XML respectivo. Esta es una desventaja que le da un mayor peso a la autenticación a medida o programada.

Esta alternativa, es fácil de implementar pero poco flexible. Sin embargo, implica no programar la seguridad de acceso a la página.

## Tipos de autenticación

Una aplicación basada en servlets puede elegir entre estos tipos de autenticación que van desde el menos seguro al más seguro:

- Básica
- Basada en el formulario
- Digest
- SSL y certificados

Se puede elegir alguno de estos métodos especificándolo en el archivo `/WEB_INF/web.xml` (de cualquier contenedor de servlets), como se muestra a continuación:

```
<web-app>
...
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Ejemplo de selección de autenticación básica</realm-name>
</login-config>
...
</web-app>
```

La autenticación básica está definida en la especificación HTTP/1.1. Cuando un usuario requiere acceder a un recurso protegido, el servidor solicita el usuario y el password. Sin embargo, este método tiene la deficiencia de que los passwords son transmitidos usando “base64 encoding” que no posee encriptación.<sup>2</sup>

La autenticación digest es similar a la básica excepto que utiliza encriptación para proteger los passwords. En realidad, la autenticación digest transmite un valor hash del password y no el password en sí.<sup>3</sup>

La autenticación basada en el formulario permite diseñar la página de acceso. Trabaja en forma similar a la autenticación básica pero utiliza una página de acceso que se muestra en lugar de una ventana de diálogo y una página de error si la identificación es negativa. Por otro lado, los passwords son transmitidos como texto puro al igual que la autenticación básica. Sin embargo, si es posible establecer autenticación segura como se explicará en el punto relativa a la seguridad del sistema, tema del presente proyecto.

---

<sup>2</sup> Cfr. HTTP/1.1 specification (<http://ftp.isi.edu/in-notes/rfc2617.txt>.)

<sup>3</sup> Cfr. HTTP/1.1 specification (<http://ftp.isi.edu/in-notes/rfc2617.txt>.)

<sup>4</sup> Cfr. SSL (<http://home.netscape.com/eng/ssl3/3-SPEC.HTM>.)

A diferencia de la autenticación digest, la autenticación basada en el formulario se define en el especificación del servlet y no la especificación HTTP.

SSL (secure sockets layer) es un mecanismo de transporte de información para asegurar su privacidad y la integridad de la data mediante encriptación. Adicionalmente, permite la verificación de la identidad del cliente y del servidor.<sup>4</sup>

El detalle de cómo añadir seguridad SSL a un servidor web, depende del servidor web para servlets, por ejemplo, Resin, Apache Tomcat, Servlet Exec y otros.

La autenticación mediante certificados de cliente está implementada en SSL y requiere que el cliente tenga un certificado de llave pública. También se puede dar seguridad a los recursos web mediante el uso de Realms, como se usa con Tomcat, los cuales dan la posibilidad de proteger un recurso con una restricción de seguridad definida y luego definir roles de seguridad que pueda acceder a este recurso.

### Autenticación programada

Programar la autenticación es una buena alternativa si se busca portabilidad y/o control total. Con este tipo de autenticación no hay dependencia con el servidor web o contenedor de servlets. Por otro lado, da algo más de trabajo que utilizar la seguridad del contenedor. Requiere la implementación de una API propia. La idea es proteger las páginas JSP con un tag customizado como este:

```
<!--Página JSP protegida -->
...
<%@ taglib='/WEB-INF/tlds/security' prefix='security' %>
...

<security:forzarLogin loginPage='/login.jsp'
                    errorPage='/error.jsp' />

<!--El resto de la página es accesada sólo si el usuario se logeo a la
sesión.
...

```

El tag **forzarLogin** busca un usuario en el ámbito de una sesión. Si el usuario está en la sesión el tag no hace nada. Si no lo está, el tag lo envía a la página de autenticación. La página de autenticación está en el atributo **loginPage**.

Si el login falla el control se envía a la página de error (error.jsp). Si el login es correcto el usuario es creado y colocado dentro de una sesión y se muestra el resto de la página.

DIAG.

La página de login envía el login a un servlet . El servlet autenticará el usuario y password y redireccionará a la página protegida o la página de error según los resultados de la autenticación.

La implementación del tag será algo similar al código que se muestra en el anexo 4.

#### ANEXO 4 (Implementación de tag de seguridad)

```
package tags;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.TagSupport;

public class ForzarLoginTag extends TagSupport {
    private String loginPag, errorPag;

    public void setLoginPag(String loginPag) {
        this.loginPag = loginPag;
    }
    public void setErrorPag(String errorPag) {
        this.errorPag = errorPag;
    }

    public int doFinTag() throws JspException {
        HttpSession session = pageContext.getSession();
        HttpServletRequest req = (HttpServletRequest)pageContext.
            getRequest();
        String pagProtegida = req.getRequestURI();

        if(session.getAttribute("usuario") == null) {
            session.setAttribute("login-pag", loginPag);
            session.setAttribute("error-pag", errorPag);
            session.setAttribute("pag-protegida", pagProtegida);

            try {
                pageContext.forward(loginPag);
                return SKIP_PAG;
            }
            catch(Exception ex) {
                throw new JspException(ex.getMessage());
            }
        }
        return TRAB_PAG;
    }
    public void release() {
        loginPag = errorPag = null;
    }
}
```

Como puede observarse con este simple tag que puede ser implementado en la página JSP que se requiera se obtiene una seguridad de acceso que retorna la página solicitada si el usuario está en la sesión, de lo contrario se le envía la página de login.

Cfr. O'Really Using Tomcat 4 Realms

<http://www.onjava.com/lpt/a/onjava/2001/07/24/tomcat.html>

Cfr. The Jakarta Project

<http://swordfish.rdfweb.org:8085/tomcat-docs/security-manager-howto.html>