

# Seguridad (44835)

## TEMA 4: Seguridad a nivel de aplicación

Dr. Raúl Peña-Ortiz

 Departamento de Informática

Universidad de Valencia

 [raul.penia@uv.es](mailto:raul.penia@uv.es)

Diciembre 2020

# Contenido

1. Introducción
2. Control de acceso
  - 2.1 Java Authorization and Autentication Service
  - 2.2 Django authentication system
3. Validación de datos de entrada
4. Programación segura
5. Referencias

# Contenido

## 1. Introducción

## 2. Control de acceso

### 2.1 Java Authorization and Autentication Service

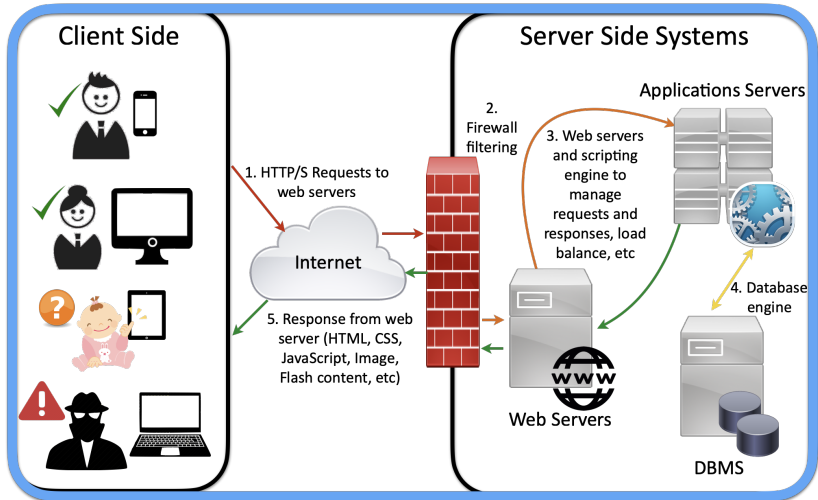
### 2.2 Django authentication system

## 3. Validación de datos de entrada

## 4. Programación segura

## 5. Referencias

# Esquema general del despliegue de aplicaciones web



# Contenido

## 1. Introducción

## 2. Control de acceso

### 2.1 Java Authorization and Autentication Service

### 2.2 Django authentication system

## 3. Validación de datos de entrada

## 4. Programación segura

## 5. Referencias

## Zonas restringidas de una aplicación web

El control de acceso de los usuarios de una aplicación web a zonas restringidas de la misma es uno de los puntos más delicados en el control de su seguridad, resolviéndose en base a dos conceptos: **autenticación** y **autorización**.

### Autenticación

Proceso para determinar si un usuario es quién dice ser. Existen dos aproximaciones típicas:

- Autenticación HTTP básica (vista en Tema 3).
- Autenticación basada en la aplicación.

### Autorización

Comprobación para saber si un usuario tiene el permiso adecuado para acceder a un cierto fichero o realizar una determinada acción, una vez que ha sido autenticado.

# Autenticación basada en aplicación

## Definición

La propia aplicación implementa un mecanismo de autenticación que implica la presentación de un formulario para que el usuario introduzca sus credenciales (login-password, certificado digital, etc) y el uso de una base de datos o directorio de usuarios para verificar la corrección de éstas.

## Ventaja

Más flexible que la autenticación HTTP básica, porque permite establecer diferentes permisos y niveles de acceso en función del usuario autenticado.

## Desventaja

Más compleja de implementar.

## Autenticación basada en aplicación (1)

### Recomendaciones al trabajar con passwords

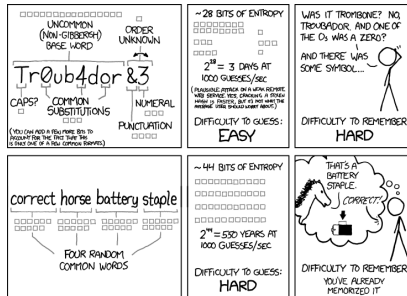
- Restringir los valores para los nombres de usuarios (email, login, dni, etc). Nombres reales suponen facilitar a los atacantes sus objetivos maliciosos.
- Almacenar los passwords de forma segura (cifrada y protegiendo el acceso a la base de datos).
- Emplear estimadores de la fortaleza de las contraseñas y exigir una puntuación mínima. Tradicionalmente se han empleado estimadores LUDS (se pide un número mínimo de letras minúsculas, mayúsculas, dígitos y/o símbolos), aunque existen otras alternativas como zxcvbn [1].
- Bloquear una cuenta cuando se detecta un número determinado de intentos incorrectos de acceso.



# Autenticación basada en aplicación (2)

## Recomendaciones al trabajar con passwords

- Forzar la actualización de claves periódicamente y mantener un histórico para evitar repeticiones en las actualizaciones.



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Fuente: <https://xkcd.com/936/>

# Autenticación basada en aplicación

## Recuperación de passwords

- Disponer de una política de recuperación de passwords en caso de olvido del usuario es esencial.
- La política puede plantear una aproximación: **automática o a través de técnicos de soporte.**
- En el caso automático hay varias estrategias:
  - Plantear durante el registro del usuario varias preguntas a las que sólo él puede responder.
  - Enviar por correo electrónico un link para introducir el nuevo password, siempre con una fecha de expiración.
  - Comunicar el password por teléfono al usuario a requerimiento del mismo, o hacer uso del móvil para cambiarlo mediante un método cruzado.
- En ambos casos deben registrarse todos los intentos de recuperación y fijar un límite de tiempo pasado el cual sería preciso recurrir al soporte técnico.

# Autenticación basada en aplicación

## Sesiones de aplicaciones web

- Una vez el usuario está autenticado, es preciso mantener esta autenticación en las posteriores peticiones HTTP.
- Para ello se recurre habitualmente al uso de **sesiones de aplicación**, que son un mecanismo que permite mantener el estado entre diferentes peticiones HTTP.
- Las sesiones funcionan de la siguiente forma:
  - Al usuario autenticado se le asigna un identificador de sesión.
  - Este identificador acompañará, de forma transparente, a cada petición del usuario garantizando que la petición proviene de un usuario previamente autenticado.
  - El identificador de sesión se suele almacenar en la propia máquina del cliente mediante una cookie, que expira al cerrar el navegador.
  - Cualquier otro dato del usuario se almacena en la sesión del servidor.

# Autenticación basada en aplicación

## Requisitos de un sistema de sesiones

- Establecer un tiempo límite de vida para la sesión.
- Regenerar el identificador de sesión cada cierto tiempo.
- Detectar intentos de ataque de fuerza bruta con identificadores de sesión.
- Requerir una nueva autenticación del usuario cuando vaya a realizar una operación importante.
- Proteger los identificadores de sesión durante su transmisión.
- Destruir la cookie al finalizar la sesión para evitar el acceso de otro usuario en un entorno público.

# Autorización

## Definición

Proceso para comprobar si un usuario tiene el permiso adecuado para acceder a una cierta información o funcionalidad de la aplicación web.

## Diseño e implementación

- El diseño e implementación del mecanismo de control de acceso en la aplicación suele realizarse usando bases de datos o directorios de usuario.
- Diseñar el mecanismo de control de acceso exige:
  - Determinar la información que será accesible por cada usuario.
  - Determinar el nivel de acceso de cada usuario a la información.
  - Especificar un mecanismo para otorgar y revocar permisos a los usuarios.
  - Proporcionar funciones a los usuarios autorizados: identificación, desconexión, petición de ayuda, cambio de contraseña, etc.
  - Ajustar los niveles de acceso a la información a la política de seguridad de la organización.

# Autorización

## Modelos de autorización

### Control de acceso discrecional

Se basa en la identidad de los usuarios o su pertenencia a ciertos grupos, de manera que el propietario de una información puede cambiar los permisos de acceso a la misma a su discreción.

### Control de acceso obligatorio

Donde cada pieza de información tiene un nivel de seguridad y cada usuario un nivel de acceso, lo cual permite determinar los permisos de acceso de cada usuario a cada pieza de información.

### Control de acceso basado en roles

Donde cada usuario tiene un rol dentro de la organización y en función de éste unos permisos de acceso.

## Taller 4.1: Aplicación Testbed

Arquitectura MVC

Autenticación por login-password

Control basado en roles

**TESTBED LOCAL**

GET STARTED TIME LOGIN REGISTER LIST OF USERS LOGOUT

LOGIN

E-mail

Password

SUBMIT RESET

Vulnerabilidades

# Contenido

## 1. Introducción

## 2. Control de acceso

### 2.1 Java Authorization and Autentication Service

### 2.2 Django authentication system

## 3. Validación de datos de entrada

## 4. Programación segura

## 5. Referencias



## Java Authorization and Authentication Service

- Java Authorization and Authentication Service (JAAS) [2] es un servicio que se encarga de gestionar la **autenticación** y la **autorización** de las aplicaciones web desarrolladas según el paradigma Java Enterprise Edition (JavaEE).
- El servicio JAAS se puede configurar declarativamente, para lo que hay que configurar una serie de elementos, tanto en el servidor de aplicaciones [S] como en la propia aplicación [A]:
  - *Security realm* [S]: el modo en el que se almacena la información relativa a los usuarios, sus contraseñas y los grupos a los que pertenecen estos usuarios.
  - La declaración de roles y su asignación a usuarios y grupos [S o A].
  - El modo que se usará para obtener la información de registro del usuario [A].
  - El rol requerido para acceder a los diferentes recursos [A].

# Java Authorization and Authentication Service

## Security Realm

- *Security Realm* [3] es el mecanismo mediante el cual el servidor almacena la información de usuarios y de grupos.

Hay varias posibilidades en función de dónde se almacena esa información:

- En un fichero (File realm).
- En una base de datos relacional (JDBC realm).
- Accediendo a LDAP (Lightweight Directory Access Protocol)
- Mediante un repositorio de certificados

# Java Authorization and Authentication Service

## Usuario/Contraseña

- El par usuario/contraseña se puede solicitar:
  - Delegando en el navegador para que muestre una ventana para recoger el usuario y contraseña.  
Esta información se puede enviar al servidor de dos formas:
    - Codificada en Base64
    - Realizando un compendio del mensaje (por ejemplo MD5) y enviándolo al servidor.
  - Mediante un formulario que envía el servidor al navegador.
- En cualquiera de los casos se debería usar el protocolo seguro HTTPS.

# Java Authorization and Authentication Service

## Autenticación web.xml

La configuración de la autenticación se realiza en el fichero web.xml:

```
<login-config>
  <auth-method> <!--Seleccionar uno-->
    BASIC| DIGEST|FORM|CLIENT-CERT
  </auth-method>
  <realm-name> <!-- Nombre que se ha dado al realm en el servidor de aplicaciones-->
    Nombre del realm
  </realm-name>

  <!-- Si es mediante formulario -->
  <form-login-config>
    <form-login-page>
      url de la pagina de login
    </form-login-page>
    <form-error-page>
      url de la pagina de error si falla el login
    </form-error-page>
  </form-login-config>
</login-config>
```

# Java Authorization and Authentication Service

## Autenticación <FORM>

Cuando se usa FORM hay que proporcionar una página que contenga un formulario donde se solicite el usuario y contraseña al usuario:

```
<!-- Pagina login.jsp -->
<html>
...
<form method="POST" action="j_security_check">
  <input type="text" name="j_username"/>
  <input type="password" name="j_password"/>
  <input type="submit" value="Enviar"/>
</form>
...
</body>
</html>
```

- `j_security_check` es el módulo que comprueba si los valores pasados en `j_username` y en `j_password` son válidos en función del *security realm* usado en la aplicación.
- `j_security_check` es un componente que proporciona JavaEE (no lo tenemos que programar nosotros).

# Java Authorization and Authentication Service

## Autenticación y `HttpServletRequest`

La interfaz `HttpServletRequest` tiene varios métodos relacionados con la autenticación:

```
// Permite realizar el login desde el código. Usa el realm configurado para la aplicación.  
// Si las credenciales no son válidas lanza ServletException  
// Este método lo podríamos usar para peticiones de autorización realizadas desde un  
dispositivo móvil  
void login(String user, String pass) throws ServletException
```

```
// Permite saber el nombre usuario o null si no está autenticado  
String getRemoteUser();
```

```
// Devuelve un java.security.Principal que contiene al usuario autenticado o null  
// Principal tiene el método getName() que devuelve el nombre de usuario  
Principal getUserPrincipal()
```

# Java Authorization and Authentication Service

## Autorización

La **autorización** regula a qué funcionalidad puede acceder un usuario en función de su rol.

Se puede conseguir de varias formas:

- De forma declarativa
- Desde el código
- Combinación declarativa y en el código

El modo de realizar la asignación de los roles a usuarios y a grupos **es dependiente del servidor de aplicaciones.**

# Java Authorization and Autentication Service

## Autorización: Roles en Tomcat

En el fichero de configuración del servidor `tomcat-users.xml` se especifica el usuario, contraseña y rol.

```
<role rolename="recursos-humanos"/>
<user username="rrhh" password="rrhhpasswd" roles="recursos-humanos" />
```

en el fichero de configuración del servidor `server.xml` se especifica el tipo de *security realm* a usar y para qué aplicación:

```
<Context path="/RRHH" docBase="RRHH">
  <Realm className="org.apache.catalina.realm.MemoryRealm" />
</Context>
```



# Java Authorization and Authentication Service

## Autorización: Roles en GlassFish

En el fichero `glassfish-web.xml` que encuentra en el directorio `WebContent/WEB-INF` de la aplicación:

```
<security-role-mapping>  
  <role-name>recursos-humanos</role-name>  
  <group-name>rrhhgroup</group-name>  
</security-role-mapping>
```

# Java Authorization and Autentication Service (1)

## Autorización: Roles en WildFly

En el fichero standalone.xml de Wildfly configuramos el Security Realm a través del jboss-cli.sh:

```
<security-domain name="webusersSEC" cache-type="default">
  <authentication>
    <login-module code="Database" flag="required">
      <module-option name="dsJndiName" value="java:jboss/datasource/webusersDS"/>
      <module-option name="principalsQuery" value="select password from users where
        username=?"/>
      <module-option name="rolesQuery" value="select role, 'Roles' from roles where
        username=?"/>
      <module-option name="hashAlgorithm" value="SHA-256"/>
      <module-option name="hashEncoding" value="base64"/>
    </login-module>
  </authentication>
  <authorization>
    <policy-module code="Database" flag="required">
      <module-option name="dsJndiName" value="java:jboss/datasource/webusersDS"/>
      <module-option name="principalsQuery" value="select password from users where
        username=?"/>
      <module-option name="rolesQuery" value="select role, 'Roles' from roles where
        username=?"/>
      <module-option name="hashAlgorithm" value="SHA-256"/>
      <module-option name="hashEncoding" value="base64"/>
    </policy-module>
  </authorization>
</security-domain>
```

## Java Authorization and Authentication Service (2)

### Autorización: Roles en WildFly

En el fichero `jboss-web.xml` que encuentra en el directorio `WebContent/WEB-INF` de la aplicación asociamos el Security Realm:

```
<jboss-web>  
  <security-domain>webusersSEC</security-domain>  
</jboss-web>
```

## Java Authorization and Authentication Service (1)

### Autorización declarativa en el fichero `web.xml`

- En el fichero `web.xml` que encuentra en el directorio `WebContent/WEB-INF` de la aplicación añadimos la información necesaria para especificar: el mecanismo que se va a utilizar para recoger la información de registro y las restricciones de acceso a los recursos del administrador y de los usuarios.
- Esta parte de la configuración es general y no depende del servidor:
  - Si el cliente solicita una URL que encaja con el patrón URL especificado y no está autenticado entonces se le pedirá la autenticación (según lo configurado en el elemento `login-config`).
  - A continuación se comprobará si pertenece al rol indicado.
  - Si no pertenece entonces se le devolverá un mensaje HTTP con código 403 indicando que no está autorizado.

## Java Authorization and Autentication Service (2)

### Autorización declarativa en el fichero web.xml

Ejemplo de autorización declarativa en el fichero web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  <display-name>Recursos humanos</display-name>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Autenticación para acceder a RRHH</realm-name>
  </login-config>
  <security-role>
    <role-name>recursos-humanos</role-name>
  </security-role>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Recursos humanos</web-resource-name>
      <url-pattern>/*</url-pattern>
      <http-method>GET</http-method>
    </web-resource-collection>
    <auth-constraint>
      <role-name>recursos-humanos</role-name>
    </auth-constraint>
    <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
</web-app>
```

# Java Authorization and Autentication Service (1)

## Autorización mediante anotaciones

También es posible configurar la autorización mediante anotaciones al declarar un componente (un Servlet o un EJB), usando:

- `@ServletSecurity` equivalente a `<security-constraint>`
- `@HTTPConstraint` para indicar todos los métodos HTTP
- `@HTTPMethodConstraint` equivalente a `<http-method>`
- `@DeclareRoles(lista de roles)` equivalente a `<security-role>`
- `@RolesAllowed (nombre del rol)` para especificar el rol que puede usar un determinado método.

## Java Authorization and Autentication Service (2)

### Autorización mediante anotaciones

Ejemplo de autorización declarativa con anotaciones [4]:

```
@WebServlet("/admin/empleados.html")
@ServletSecurity(@HttpConstraint(transportGuarantee = TransportGuarantee.CONFIDENTIAL,
                                rolesAllowed={"recursos_humanos"}))
public class Empleados extends HttpServlet{
    ...
}
```

```
@WebServlet("/empleados/nominas.html")
@ServletSecurity(
    httpMethodConstraints={
        @HttpMethodConstraint("GET",rolesAllowed={"empleados","recursos_humanos"},
                               transportGuarantee=TransportGuarantee.CONFIDENTIAL),
        @HttpMethodConstraint("POST",rolesAllowed={"registered_users"},transportGuarantee=
                               TransportGuarantee.CONFIDENTIAL)
    }
}
public class Productos extends HttpServlet{
    ...
}
```

# Java Authorization and Autentication Service

## Autorización desde el código

El siguiente método está declarado en la clase `HttpServletRequest`:

```
// Comprueba si el usuario autenticado pertenece a un determinado rol  
boolean isUserInRole(String rol)
```

Ejemplo:

```
@WebServlet("/empleados/nominas.html")  
@DeclareRoles("empleados")  
public class Productos extends HttpServlet{  
    public void doGet(HttpServletRequest req, HttpServletResponse resp)  
        throws ...{  
  
        if (req.isUserInRole("empleados")){  
            // Acción  
        }  
  
    }  
}
```



# Contenido

## 1. Introducción

## 2. Control de acceso

### 2.1 Java Authorization and Autentication Service

### 2.2 Django authentication system

## 3. Validación de datos de entrada

## 4. Programación segura

## 5. Referencias

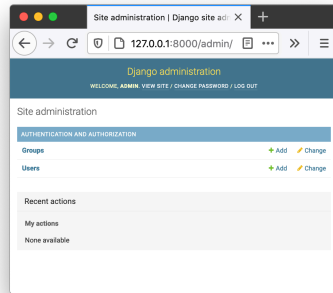
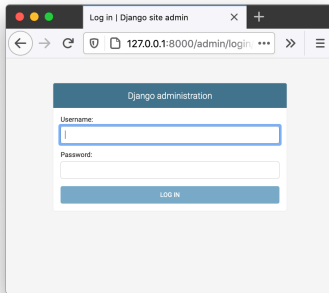
## Django authentication system

- Django proporciona un sistema de autenticación y autorización que le permite verificar credenciales de usuario y definir que acciones puede realizar cada usuario.
- El framework incluye modelos para Users y Groups (una forma genérica de aplicar permisos a más de un usuario a la vez), permisos/indicadores (permissions/flags) que designan si un usuario puede realizar una tarea, formularios y vistas para iniciar sesión en los usuarios, y view tools para restringir el contenido.
- Esta sección introduce conceptos básicos del framework y remite al tutorial de referencia [5] o a la documentación oficial [6] para profundizar.

# Django authentication system

## Usuarios y Grupos

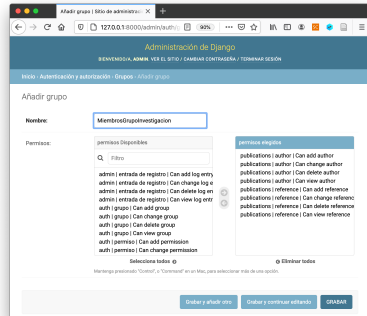
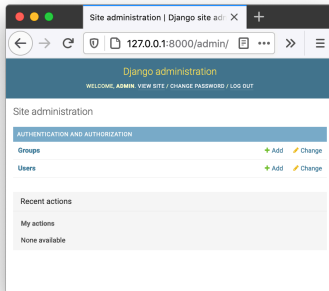
La creación de Users y Groups la podemos hacer a través de la interfaz gráfica de administración (<http://127.0.0.1/admin>) o a través de la consola Django.



# Django authentication system

## Creación de un grupo

Si tomamos como referencia la aplicación Django del Taller 9 de la asignatura *Programación del lado del servidor (44825)*, en la creación del grupo `MiembrosGrupoInvestigacion` podemos añadir permisos para crear Autores y Publicaciones.



# Django authentication system

## Creación de un usuario de un grupo

Añadir usuario | Sitio de administrac X +

127.0.0.1:8000/admin/a 50%

### Administración de Django

BENVENID@A ADMIN. VER EL SITIO / CAMBIAR CONTRASEÑA / TERMINAR SESIÓN

Inicio · Autenticación y autorización · Usuarios · Añadir usuario

#### Añadir usuario

Primero introduzca un nombre de usuario y una contraseña. Luego podrá editar el resto de opciones del usuario.

**Nombre de usuario:**

Requiere: 150 caracteres como máximo. Únicamente letras, dígitos y @/./+/\_/.

**Contraseña:**

Su contraseña no puede ser idéntica tanto a su otro información personal.  
 Su contraseña debe contener al menos 8 caracteres.  
 Su contraseña no puede ser una clave utilizada comúnmente.  
 Su contraseña no puede ser completamente numérica.

**Contraseña (confirmación):**

Para verificar, introduzca la misma contraseña anterior.

Modificar usuario | Sitio de adminis X +

127.0.0.1:8000/admin/auth/us: ...

### Administración de Django

BENVENID@A ADMIN. VER EL SITIO / CAMBIAR CONTRASEÑA / TERMINAR SESIÓN

Inicio · Autenticación y autorización · Usuarios · Pablo

#### Modificar usuario

**Nombre de usuario:**

Requiere: 150 caracteres como máximo. Únicamente letras, dígitos y @/./+/\_/.

**Contraseña:**

Su contraseña no puede ser idéntica tanto a su otro información personal.  
 Su contraseña debe contener al menos 8 caracteres.  
 Su contraseña no puede ser una clave utilizada comúnmente.  
 Su contraseña no puede ser completamente numérica.

**Información personal**

**Nombre:**

**Apellido:**

**Dirección de correo electrónico:**

**Permisos**

☒ **Active**  
 Indica si el usuario debe ser tratado como activo. Desmarcar este indicador impide la forma de acceso.

☐ **Is staff**  
 Indica si el usuario puede editar en estado de administración.

☐ **Is superuser**  
 Indica si el usuario puede tener todos los permisos de administrador.

**Grupos:**

**grupos disponibles**

**grupos elegidos**

**grupos de usuario elegidos**

**Permisos de usuario:**

**permisos de usuario disponibles**

**permisos de usuario elegidos**

**Notas importantes**

**Última fecha de sesión:**

**Fecha de:**

**Notas de:**

# Django authentication system (1)

## Vistas de autenticación

Adición de las URLs de autenticación al final del fichero de `taller9/publications/urls.py`:

```
...  
urlpatterns += [  
    path('accounts/', include('django.contrib.auth.urls')),  
]  
...
```

## Django authentication system (2)

### Vistas de autenticación

#### Creación de la plantilla de autenticación

taller9/publications/templates/registration/login.html, asociada a la URL <http://127.0.0.1:8000/accounts/login/>:

```
{% block content %}
{% if form.errors %}
<p>Error en login y password, inténtelo otra vez.</p>
{% endif %}
{% if next %}
{% if user.is_authenticated %}
<p>No tienes permisos de acceso.</p>
{% else %}
<p>Debes autenticarte para acceder a esta página.</p>
{% endif %}
{% endif %}
<form method="post" action="{% url 'login' %}">
  {% csrf_token %}
  <div><td>{{ form.username.label_tag }}</td><td>{{ form.username }}</td></div>
  <div><td>{{ form.password.label_tag }}</td><td>{{ form.password }}</td></div>
  <div>
    <input type="submit" value="login" />
    <input type="hidden" name="next" value="{{ next }}" />
  </div>
</form>
{% endblock %}
```

## Django authentication system (3)

### Vistas de autenticación

Creación de la plantilla de cierre de sesión

taller9/publications/templates/registration/logged\_out.html,  
asociada a la URL `http://127.0.0.1:8000/accounts/logout/`:

```
{% block content %}
<p>Sesión cerrada !</p>

<a href="{% url 'login' %}">Pincha para autenticarte.</a><br/>
<a href="{% url 'index' %}">Pincha para ir a home.</a>
{% endblock %}
```



## Django authentication system (4)

### Vistas de autenticación

Autorizar el acceso a una vista, si el usuario está autenticado y pertenece al grupo creado, modificando la vista taller9/publications/views.py:

```
...
from django.contrib.auth.decorators import login_required
from django.contrib.auth.decorators import user_passes_test

@login_required
@user_passes_test(lambda u: u.groups.filter(name='MiembrosGrupoInvestigacion').exists())
def listado_autores(request):
    request.session['menu_option'] = 2
    authors = Author.objects.all().order_by('surnames')
    context = {'authors_list': authors,
               'body': 'autores.html'}

    return render(request, 'template.html', context)
...
```

## Django authentication system (5)

### Vistas de autenticación

Añadir al `taller9/templates/menu.html`, si el usuario está autenticado, opción de cerrar la sesión:

```
...  
{% if user.is_authenticated %}  
<div class="vertical-menu">  
    <a href="#" class="titulo">Usuario</a>  
    <a href="{% url 'logout' %}">Cerrar sesión</a>  
</div>  
{% endif %}  
...
```

# Contenido

## 1. Introducción

## 2. Control de acceso

### 2.1 Java Authorization and Authentication Service

### 2.2 Django authentication system

## 3. Validación de datos de entrada

## 4. Programación segura

## 5. Referencias

## El origen de las vulnerabilidades de las aplicaciones web

- El problema más frecuente que presentan las aplicaciones web se encuentra en la validación inapropiada de los datos de entrada.
- Este hecho es el causante de algunas de las vulnerabilidades más importantes vistas en el Tema 1 (Inyección SQL, Cross-Site Scripting, Buffer Overflow, etc).
- Aspectos a considerar son:
  - Fuentes de entrada:
    - Cadenas URL.
    - Cookies.
    - Cabeceras HTTP.
    - Campos de formularios.
  - Inyección de datos.
  - Estrategias de protección.

## Fuentes de entrada

### Cadenas URL

- Si se envía un formulario con el método GET, los nombres y valores de todos los elementos del formulario aparecen detrás de la URL de la página invocada:

`https://testbed.local/login.html?P_LOGIN=AAA&P_PASSWORD=BB`

de manera que es muy fácil modificar esta cadena:

`https://testbed.local/login.html?P_LOGIN=NN&P_PASSWORD=PP`

- La aplicación debe comprobar el valor recibido aunque proceda de una lista desplegable con unos valores predefinidos, ya que el usuario ha podido modificar manualmente la URL.
- Este problema se da también en los enlaces que incluyen parámetros.
- Siempre que se envíen datos sensibles hay que acompañarlos de un identificador de sesión y comprobar que el usuario asociado a la sesión tiene acceso a la información requerida.

## Fuentes de entrada

### Cookies

- Método habitual para mantener el estado o almacenar preferencias del usuario.
- Pueden ser modificadas por el cliente para engañar al servidor, de manera que el peligro dependerá de lo que se almacene en la cookie.

Por ejemplo, la siguiente cookie:

```
Cookie: lang=es-ES; ADMINISTRADOR=no; time=12:30GMT;
```

puede ser modificada fácilmente por:

```
Cookie: lang=es-ES; ADMINISTRADOR=yes; time=12:30GMT;
```

para obtener privilegios de administrador.

### Recomendación

No almacenar en la cookie nada más que el identificador de sesión, manteniendo la información relevante en el servidor.

## Fuentes de entrada

### Cabeceras HTTP

- Las cabeceras HTTP contienen información de control enviadas entre el cliente y el servidor. Por ejemplo:

```
Host: testbed.local
Pragma: no-cache
Cache-Control: no-cache
User-Agent: ...
Referer: ...
Content-type: text/html; charset=UTF-8
Content-length: ...
```

- Si la aplicación web utiliza las cabeceras recibidas del cliente, hay que tener en cuenta que éstas pueden haber sido manipuladas, por lo que deben ser verificadas para evitar inyección de datos no deseados. Por ejemplo:

```
Accept-Language: es
```

podría ser modificada por:

```
Accept-Language: select * from ...
```

## Fuentes de entrada (1)

### Campos de formularios

- Los formularios pueden ser modificados para enviar lo que el usuario desee. Basta con guardar la página, modificar el código y recargarlo en el navegador. Las limitaciones impuestas en el propio formulario se pueden saltar perfectamente.

Por ejemplo:

```
<input type="text" name="campo" maxlength="100">
```

podría modificarse así:

```
<input type="text" name="campo" maxlength="100000">
```

con el consiguiente riesgo para la aplicación si el valor no se chequea adecuadamente.



## Fuentes de entrada (2)

### Campos de formularios

- Los campos ocultos (HIDDEN) también son vulnerables a este ataque, por lo que no deben utilizarse para almacenar información sensible.

Por ejemplo:

```
<input name="tarjetapago" type="hidden" value="XXXXXXXX">  
<input name="precio" type="hidden" value="199.99">
```

### Recomendación

Utilizar sesiones y almacenar la información sensible en el servidor es mucho más seguro.

# Inyección de datos

## Idea

Consiste en inyectar en la aplicación datos introducidos por el usuario.

## Ejemplo

La siguiente query:

```
SELECT * FROM productos WHERE id=?
```

se convierte en:

```
SELECT * FROM productos WHERE id=225
```

al hacer click sobre el producto de interés para el usuario (id=255).

# Inyección de datos

## Inyección maliciosa

### Objetivo

Suministrar datos que al ser inyectados en la aplicación causen un efecto dañino.

### Estructura del ataque

El ataque está completamente contenido en los datos suministrados por el atacante, que se pueden dividir en tres partes:

- **Conector de prefijo:** ayuda a encajar el ataque en la aplicación.
- **Payload:** contiene el ataque.
- **Conector de sufijo:** ayuda a encajar el ataque en la aplicación.

# Inyección de datos

## Facilidades para la inyección maliciosa

- Los ataques de inyección de datos requieren de un amplio conocimiento del lenguaje, las herramientas y las técnicas utilizadas en la aplicación web.
- La elección de los conectores puede verse facilitada por factores como:
  - El acceso al código fuente (proyectos open source).
  - Los mensajes de error detallados.
  - El uso de técnicas de programación conocidas en el desarrollo de las aplicaciones.

# Inyección de datos

## Inyección SQL

Si la siguiente query:

```
SELECT * FROM usuarios WHERE login = ? AND password = ?
```

se usa en una autenticación de aplicación basada en login y password, podemos inyectar un ataque en el campo login con los siguientes componentes:

- Prefijo: 'OR
- Payload: 1=1
- Sufijo: OR login='

obteniendo la query:

```
SELECT * FROM usuarios WHERE login = '' OR 1=1 OR login = '' AND password = ''
```

que nos devolverá todos los usuarios de la tabla.

# Inyección de datos

## Inyección de órdenes del Sistema Operativo

- Casi todos los lenguajes de programación disponen de funciones que permiten la ejecución de órdenes del Sistema Operativo.
- Estas funciones son útiles (e.g., para el manejo de ficheros o el envío de correo), pero plantean serios riesgos ya que se pueden manipular para:
  - Modificar las órdenes ejecutadas.
  - Modificar los parámetros pasados a las órdenes del sistema.
  - Ejecutar órdenes adicionales.
- Por ejemplo, el siguiente código PHP:

```
system ("ls $dir");
```

mostrará el fichero de passwords del sistema cuando la variable \$dir adopte el valor:

```
"/tmp; cat /etc/passwd"
```

# Inyección de datos

## Inyección HTML - Cross-Site Scripting (CSS)

- Consiste en insertar en una aplicación web (e.g., un CMS o un LMS) un contenido con código malicioso (e.g., JavaScript), de manera que cuando otro usuario visualice el contenido o interactúe con el mismo el código se ejecutará en su máquina con fines maliciosos.
- Este tipo de ataques puede usar:
  - Marcas como `<SCRIPT>`, `<A>`, `<IMG>` o `<IFRAME>`, de manera que el ataque se produce cuando el texto se visualiza en el navegador de otro usuario.
  - Eventos, como *ONCLICK*, asociados habitualmente a elementos de formulario.

## Estrategias de protección

- Aceptar únicamente datos válidos conocidos, es la estrategia más adecuada ya que se limita en general a validar:
  - Tipo de dato.
  - Longitud máxima y mínima.
  - Datos obligatorios.
  - Si hay una lista enumerada de posibles valores, comprobar que está en ella.
  - Si hay un formato o plantilla (e.g., expresión regular) específico, comprobar que lo cumple.
  - Si es texto libre, que sólo contiene caracteres válidos.
  - Si se permiten caracteres peligrosos: convertirlos.
  - Los valores se considerarán por separado y luego en conjunto cuando lo requieran (e.g., cuando forman queries SQL o tienen relaciones semánticas entre sí).
- Rechazar datos no válidos conocidos, es difícil de aplicar porque implica conocer todos los datos peligrosos.
- Sanear todos los datos consiste en transformar los datos en una representación que no sea un riesgo. Por ejemplo, transformar "<" en "&lt;".
- Además, se deben utilizar las funciones de biblioteca en lugar de invocar órdenes del sistema operativo, en la medida de lo posible.



# Contenido

## 1. Introducción

## 2. Control de acceso

### 2.1 Java Authorization and Autentication Service

### 2.2 Django authentication system

## 3. Validación de datos de entrada

## 4. Programación segura

## 5. Referencias

## Inicialización de variables (1)

### Recomendación

En general es recomendable inicializar todas las variables antes de usarlas.

### Vulnerabilidad por no inicialización de variables

Existen lenguajes de programación como PHP que permite la creación de variables globales a partir del entorno, las cookies, la información del servidor, y los parámetros de los formularios, lo que puede suponer una vulnerabilidad.

## Inicialización de variables (2)

### Ejemplo vulnerabilidad

Sea el siguiente código de la página `check.php`

```
if (check_user()) $admin = true;
```

una llamada de la forma

```
check.php?admin=1
```

permitiría obtener privilegios de administrador.

### Solución

Dar un valor inicial a la variable:

```
$admin = false;  
if (check_user()) $admin = true;
```

## Gestión de errores

- Los mensajes de error son una fuente de información muy útil para los atacantes, ya que pueden proporcionar información sensible que les facilite sus ataques.
- En un entorno de producción debe evitarse la aparición de mensajes de aviso o error.
- Los errores deben volcarse en un fichero especificado en lugar de en la pantalla.

## Protección de información

- Toda información sensible de nuestra aplicación web (e.g, ficheros configuración) debe almacenarse por separado del programa que la utiliza y preferentemente en un directorio situado fuera del árbol de directorios de la web para evitar que pueda ser accedida por su URL (e.g., dentro de WEB-INF en J2EE).
- Evitar en el código comentarios que proporcionen demasiados detalles acerca del funcionamiento de la aplicación. Eliminarlos de la versión de producción.
- Eliminar las órdenes de depuración colocadas en el código durante su desarrollo: uso de librerías para la gestión de logs (e.g., log4j).
- Proteger los ficheros que tengan el acceso restringido.

# Contenido

## 1. Introducción

## 2. Control de acceso

### 2.1 Java Authorization and Autentication Service

### 2.2 Django authentication system



## 3. Validación de datos de entrada

## 4. Programación segura

## 5. Referencias

## Referencias (1)

- [1] Daniel Lowe Wheeler. Zxcvbn: Low-budget password strength estimation. In *Proceedings of the 25th USENIX Conference on Security Symposium, SEC'16*, page 157–173, USA, 2016. USENIX Association. ISBN 9781931971324. URL [https://www.usenix.org/system/files/conference/usenixsecurity16/sec16\\_paper\\_wheeler.pdf](https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_wheeler.pdf).
- [2] Java Authentication and Authorization Service (JAAS) Reference Guide. <https://docs.oracle.com/javase/7/docs/technotes/guides/security/jaas/JAASRefGuide.html>, 1993-2020. Accedido en Diciembre de 2020.
- [3] Java Platform, Enterprise Edition: The Java EE Tutorial - 47.5 Working with Realms, Users, Groups, and Roles. <https://docs.oracle.com/javaee/7/tutorial/security-intro005.htm>, 2014. Accedido en Diciembre de 2020.
- [4] Shing Wai Chan. Follow up on Servlet 3.0 Security Annotations. <https://blogs.oracle.com/swchan/follow-up-on-servlet-30-security-annotations>, 2009. Accedido en Diciembre de 2020.
- [5] Mozilla and individual contributors. Tutorial de Django Parte 8: Autenticación y permisos de Usuario. <https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Authentication>, July 2020. Accedido en Diciembre de 2020.
- [6] Python Software Foundation. Django documentation: Using the Django authentication system. <https://docs.djangoproject.com/en/3.1/topics/auth/default/>, 2005-2020. Accedido en Diciembre de 2020.

© 2020 Dr. Raúl Peña-Ortiz  
 Departamento de Informática  
Universidad de Valencia  
 raul.pena@uv.es