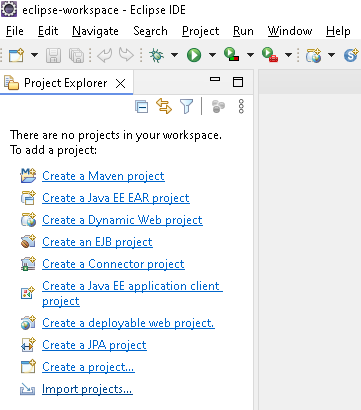
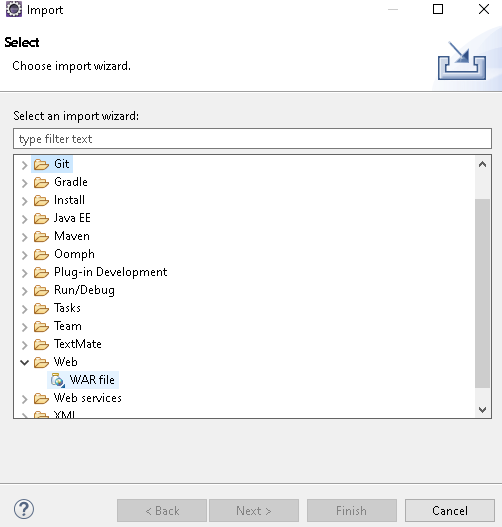
**Proyecto 1 Inyección por SQL**

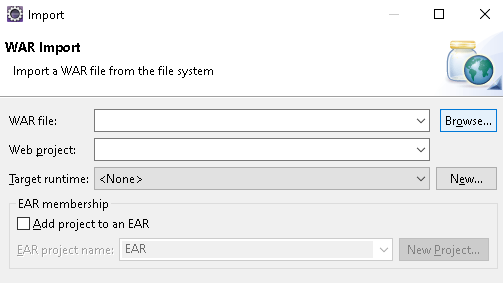
Importar el proyecto

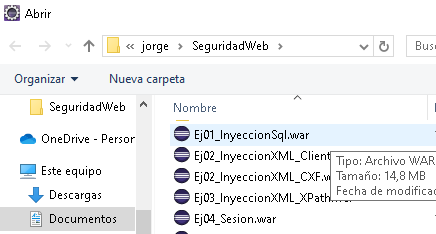


Seleccionar web -> WAR file

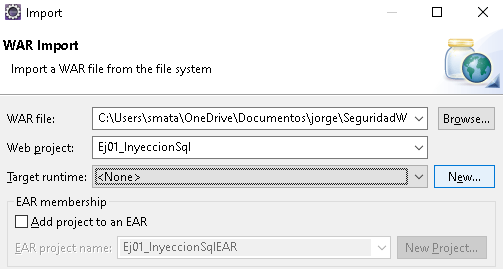


Luego, buscamos con el botón de Browse

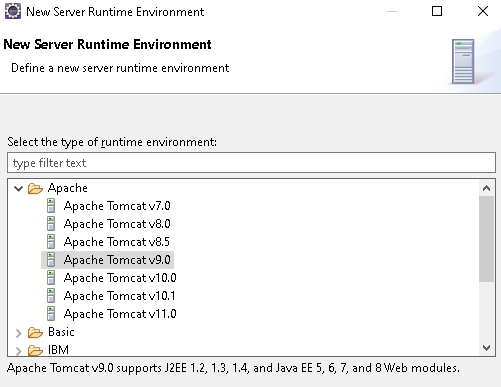


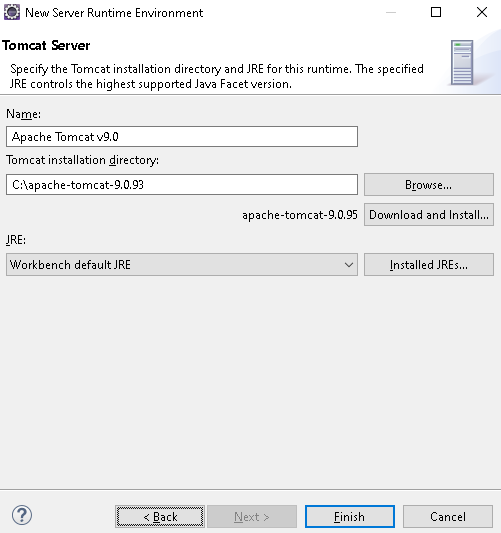


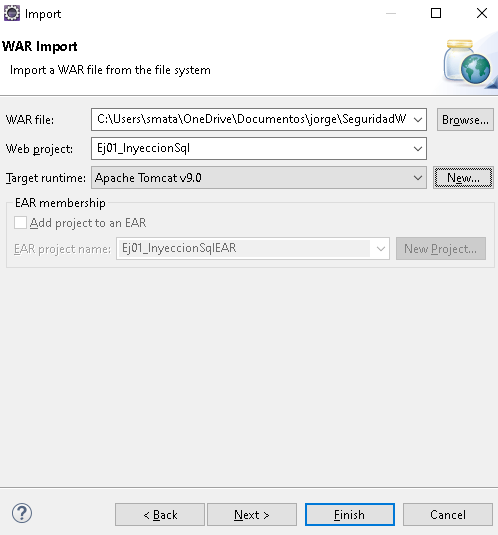
Ahora debemos seleccionar el servidor con el combo Target Runtime -> new



Si tenemos instalado Apache Tomcat de antemano podemos seleccionar su ubicación, sino también tenemos la posibilidad de instalarlo desde el propio Eclipse.

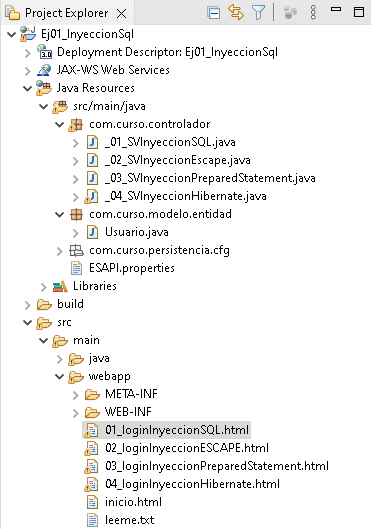






**Inyección SQL**

Una vez abierto el proyecto, levantamos el primer ejercicio 01\_loginInyeccionSQL.html



**Nota**: en la carpeta lib de WEB-INF agregar el driver de MySQL desde

<https://dev.mysql.com/downloads/connector/j/>

Y descargar el mysql-connector-java-9.2.0.jar

Además, debe estar la dependencia declarada en el archivo pom.xml

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>9.2.0</version>

</dependency>

*\_01\_SVInyeccionSQL.java*

@Override

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

String login = request.getParameter("login");

String pw = request.getParameter("pw");

Connection cx = **null**;

Statement st = **null**;

ResultSet rs = **null**;

**try** {

// Registrar el driver MySQL

Class.*forName*("com.mysql.cj.jdbc.Driver");

// Intentar establecer la conexión

cx = DriverManager.*getConnection*(

"jdbc:mysql://localhost:3306/bbdd",

"root",

"1234"

);

// Si llegamos aquí, la conexión fue exitosa

System.***out***.println("¡Conexión exitosa!");

// Crear statement y ejecutar consulta

st = cx.createStatement();

String sql = "select \* from usuario where login='" + login + "' and pw='" + pw + "'";

System.***out***.println("Consulta: " + sql);

rs = st.executeQuery(sql);

**if**(rs.next()){

response.sendRedirect("inicio.html");

} **else** {

response.sendRedirect("01\_loginInyeccionSQL.html");

}

} **catch** (SQLException e) {

// Manejo específico para errores de SQL

System.***err***.println("Error al conectar o consultar la base de datos:");

System.***err***.println(e.getMessage());

response.sendError(HttpServletResponse.***SC\_INTERNAL\_SERVER\_ERROR***,

"Error en la base de datos");

} **catch** (ClassNotFoundException e) {

// Manejo específico para errores del driver

System.***err***.println("Error: El driver MySQL no está registrado:");

System.***err***.println(e.getMessage());

response.sendError(HttpServletResponse.***SC\_INTERNAL\_SERVER\_ERROR***,

"Error en la configuración del servidor");

} **finally** {

// Cerrar recursos en orden inverso

**try** {

**if** (rs != **null**) rs.close();

**if** (st != **null**) st.close();

**if** (cx != **null**) cx.close(); // Ahora cx nunca será null aquí

} **catch** (SQLException e) {

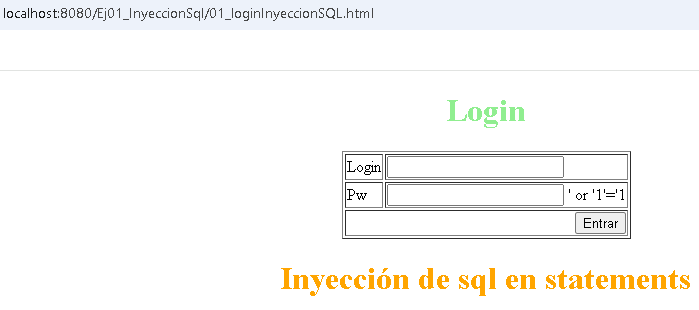
System.***err***.println("Error al cerrar recursos:");

System.***err***.println(e.getMessage());

}

}

}



*Probemos con los siguientes pares de valores*

select \* from usuario where login='a' and pw='a'

select \* from usuario where login='b' and pw='b'

select \* from usuario where login='cualquier cosa' and pw='cualquier cosa'

select \* from usuario where login='grgr' and pw='' or '1'='1' **Aquí se produce la inyección SQL**

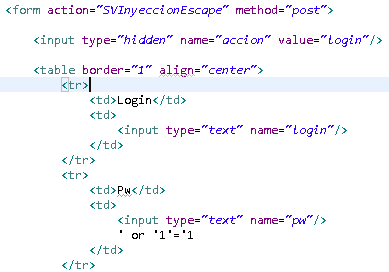
Este código es vulnerable porque el atacante puede ingresar cualquier cosa después de los apóstrofes, incluyendo comandos SQL maliciosos. La condición or '1'='1' siempre será verdadera, permitiendo acceso a los datos de los usuarios sin credenciales válidas.

**Inyección: Sanitizar la entrada por caracteres de escape**

Una de las primeras soluciones para intentar evitar una inyección de SQL es introducir caracteres de escape dentro de la sentencia SQL que nos han introducido o que nos han intentado inyectar en el formulario, lo que significaría sanear la entrada.

Para que nuestro motor de base de datos no entienda las sentencias SQL que ha intentado introducir un hacker, tenemos que las cadenas que nos han introducido en el formulario y pasarla por alguna librería hecha por nosotros o por algún tercero, a fin de evitar que nuestra base de datos "no entienda" esas sentencias SQL que nos están enviando.

Ejecutemos el ejemplo 02\_loginInyeccionEscape.html, cuya salida se dirige al servlet correspondiente en el controlador de la aplicación. El código del formulario es idéntico al primer ejemplo.



Texto

El contenido generado por IA puede ser incorrecto.

Estamos utilizando una librería de terceros, en este caso ESAPI (Enterprise Security API) de OWASP para proteger contra inyecciones SQL mediante varios niveles de codificación y validación.

Validación inicial por ESAPI

* Verifica que los datos sean del tipo esperado
* Comprueba que no contengan patrones maliciosos comunes
* Valida la longitud y formato de los campos

Codificación con MySQLCodec

Convierte caracteres especiales en secuencias seguras:

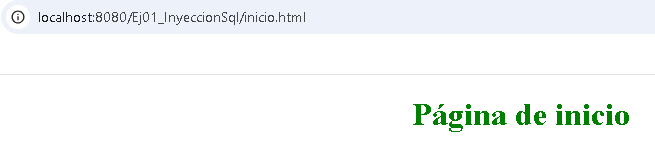
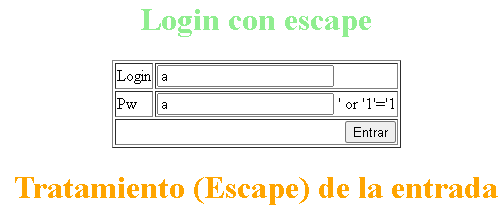
* Las comillas simples (') se convierten en dos comillas simples ('')
* Los puntos y comas (;) se escapan para evitar múltiples consultas
* Los comentarios (--) se neutralizan para impedir saltos de código

Ejecución Segura

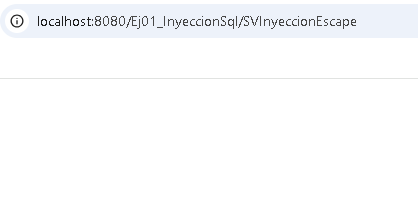
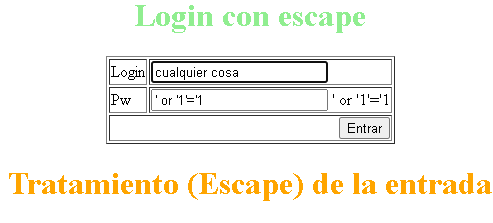
* La consulta resultante es tratada como datos literales
* El motor de base de datos interpreta los caracteres escapados correctamente
* Intentos de inyección SQL son neutralizados automáticamente

Por ejemplo, si un atacante intenta ingresar admin'; DROP TABLE users; -- como login, ESAPI lo convertirá en una cadena segura que el motor de base de datos interpretará como un valor literal, no como código SQL ejecutable.

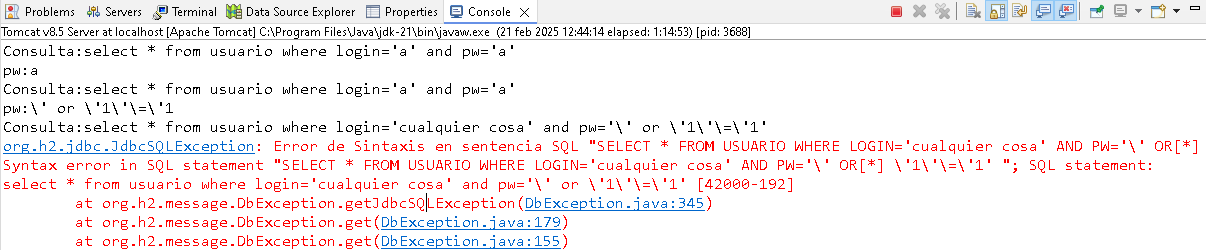
Al ejecutar el archivo 02\_loginInyeccionEscape.html e ingresar valores correctos de user y pass:



Pero si ahora ingresamos una inyección sql, el servlet \_02\_SVInyeccionEscape captura el código malicioso y lo neutraliza, de modo tal que se queda la respuesta en el servlet sin pasar a la base de datos.



Además, se produce una excepción, por lo que, podemos monitorizar en qué momento nos están intentando hackear.



**Inyección: Sanitizar la entrada mediante PreparedStatement**

Otra solución factible para evitar el SQL Injection es la introducción de las sentencias preparadas, es decir, hacer un tratamiento de todo lo que estamos accediendo a partir de nuestro formulario para producir la sentencia SQL en concreto, sin la concatenación.

Lo que cambiaremos es que este caso es que tanto el login como el password no van a ser concatenados dentro de la sentencia SQL sino que vamos a generar una sentencia SQL por medio de parámetros de entrada.

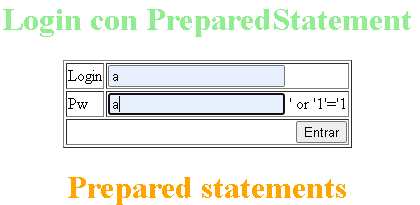
En este caso no se genera ninguna excepción pero como contrapartida no sabemos cuándo nos están intentando hackear y no estamos haciendo ninguna monitorización.

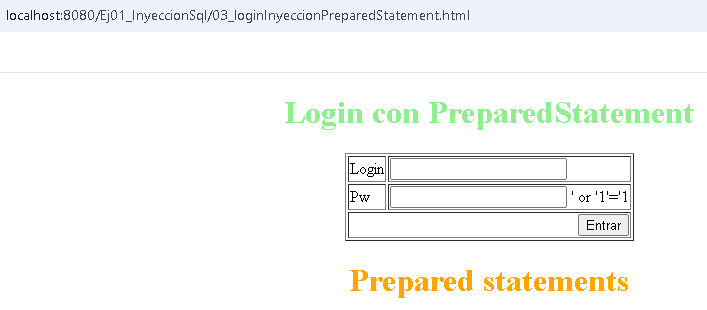
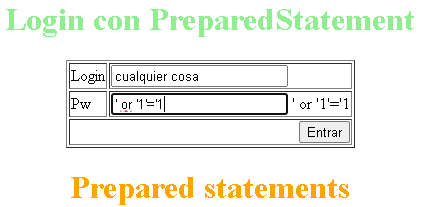
Vamos a ejecutar el archivo 03\_loginInyeccionPreparedStatement.html que intentara enviar los datos al servlet correspondiente.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Y ahora vemos la entrada de datos y la salida para cada caso





Podemos observar que en el caso de inyección SQL, el código malicioso no pasa a la base de datos y se re direcciona a la pagina HTML de origen del formulario.