

# Sixto Jansa Sanz,  
# Jorge Utrilla Olivera,  
# y Jose Miguel Maldonado Del Pozo  
# declaramos que esta solución es fruto exclusivamente de 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.

# Vulnerabilidades de El Coladero

Informe de vulnerabilidad
Ruta(s) de la aplicación involucrada(s)
POST /insert_question, /insert_reply
Tipo de vulnerabilidad
XSS persistente
Situaciones peligrosas o no deseadas que pueden provocar
<p>Inclusión de etiquetas HTML, con atributos "src=" a páginas maliciosas en las que descargar archivos infectados.</p> <p>Ejecuciones de código javascript indeseadas por parte del usuario y de la aplicación, estas ejecuciones podrían modificar todo el árbol HTML de la página para que se muestre de forma distinta, o realizar peticiones contra un servidor con fines de ataque de denegación de servicio distribuido, también podrían redirigir al usuario a una página maliciosa en la que llevarle a un ataque más complejo, en definitiva, cualquier cosa que se pueda realizar con javascript.</p> <p>En El Coladero no hay sesiones ni autenticación del usuario, pero desde javascript se podría capturar las cookies e enviarlas a un servidor malicioso, realizando así un secuestro de sesión (session hijacking).</p>
Ejemplo paso a paso de cómo explotar la vulnerabilidad
<p>Se introducen los datos de la pregunta y en uno de los campos se añade el script que se va a guardar en la base de datos.</p> <p>Autor: <input type="text" value="jake"/></p> <p>Título: <input type="text" value="mate"/></p> <p>Etiquetas: <input type="text" value="hehe"/></p> <p><input type="text" value="&lt;script&gt;alert( 'jakeado' )&lt;/script&gt;"/></p> <p>Cuerpo: <input type="text"/></p> <p><input type="button" value="Preguntar"/></p> <p>Al cargar el cuerpo de dicha pregunta el script se ejecuta automáticamente, mostrando el</p>

mensaje:

## El Coladero

### Foro de preguntas y respuestas

Búsqueda por etiqueta:

Título: **mate**

Autor: jake

Fecha: 2017-01-26 21:43:41

Etiquetas: hehe

Cuerpo:

jakeado

#### Medidas para mitigar la vulnerabilidad

- Se debería comprobar la longitud y caracteres de los campos, por ejemplo:
  - autor: sólo los caracteres [a-z][A-Z][0-9], '-', '\_' , con longitud mínima de 3 y longitud máxima de 10
  - título: igual que autor pero con long. máxima de 20
  - etiqueta: sólo los caracteres [a-z][A-Z][0-9] y long. máxima de 200
  - cuerpo: longitud mínima de 5 y máxima de 2000
- Se deberían escapar los caracteres del cuerpo de la pregunta, para que se muestren caracteres como '>' pero sin interferir en el HTML de la web.

Informe de vulnerabilidad
Ruta(s) de la aplicación involucrada(s)
GET /search_question
Tipo de vulnerabilidad
XSS reflejado
Situaciones peligrosas o no deseadas que pueden provocar
<p>Al igual que el XSS persistente, permite la inclusión de HTML o código javascript no deseado por parte del usuario y la aplicación, con todo lo que conlleva.</p> <p>En este caso, no se vería reflejado en todos los usuarios de la página web, sino sólo en los que accedieran con el enlace envenenado, ya que no se guarda en la base de datos.</p>
Ejemplo paso a paso de cómo explotar la vulnerabilidad
<p><b>Búsqueda por etiqueta:</b> <input type="text" value="&lt;script&gt;alert('hello')&lt;/script&gt;"/> <input type="button" value="Buscar"/></p> <p>al introducir ese script y darle a buscar, hará la petición GET          /search_question?tag=%3Cscript%3Ealert%28%22hello%22%29%3C%2Fscript%3E en          la que posteriormente se mostrará el valor de tag como valor buscado, ejecutándose así el código          javascript deseado. Si se pasase el enlace  <a href="http://localhost:8080/search_question?tag=%3Cscript%3Ealert%28%22hello%22%29%3C%2Fscript%3E">http://localhost:8080/search_question?tag=%3Cscript%3Ealert%28%22hello%22%29%3C%2Fscript%3E</a>          a otra persona, esta sería víctima del ataque.</p>
Medidas para mitigar la vulnerabilidad
<ol style="list-style-type: none"> <li>1. Permitir sólo los caracteres posibles de las etiquetas, es decir, [a-z][A-Z][0-9] y una longitud máxima de 20.</li> </ol>

Informe de vulnerabilidad
Ruta(s) de la aplicación involucrada(s)
POST /insert_question
Tipo de vulnerabilidad
SQL injection
Situaciones peligrosas o no deseadas que pueden provocar
Podría realizar cualquier acción sobre la base de datos, ya sea insertar nuevos campos, eliminar otros, borrar tablas o incluso la base de datos completa.
Ejemplo paso a paso de cómo explotar la vulnerabilidad
<div> <div>Autor: <input type="text" value="inyectate"/></div> <div>Título: <input type="text" value="esta"/></div> <div>Etiquetas: <input type="text" value="jeje"/></div> <div> <input type="text" value="', ''); DROP Table Questions; --"/> </div> <div>Cuerpo: <input type="text"/></div> <div>Preguntar</div> </div> <p>Al usarse la función <code>executescript</code> de <code>sqlite3</code> se permite la ejecución de varias sentencias SQL a la vez, por lo que introduciendo lo anterior, se consigue terminar la consulta legítima con <code>', '');</code> y posteriormente se procede a ejecutar la sentencia deseada sobre la base de datos, por último se comenta con <code>--</code> lo último de la consulta legítima.</p> <p>Quedando así la consulta completa:</p> <pre>INSERT INTO Questions(author, title, tags, body, time) VALUES ('jake', 'mate', 'hehe', '', ''); DROP Table Questions; --', CURRENT_TIMESTAMP)</pre>
Medidas para mitigar la vulnerabilidad
<ol style="list-style-type: none"> <li>1. Sólo se debería permitir la ejecución de una sentencia SQL usando la función <code>execute()</code>.</li> <li>2. Se deberían escapar los caracteres, sobre todo las comillas simples, usando los argumentos de la función <code>execute</code>, ya que esta se encarga de sanitizar los datos ante posibles inyecciones SQL.</li> <li>3. Se debería crear un nuevo usuario en la base de datos con permisos exclusivamente para insertar en la tabla <code>Questions</code> y no para borrar filas o tablas enteras.</li> </ol>

Informe de vulnerabilidad
Ruta(s) de la aplicación involucrada(s)
POST /show_question
Tipo de vulnerabilidad
SQL injection
Situaciones peligrosas o no deseadas que pueden provocar
Puede revelar información sensible de la base de datos, como campos que no deberían mostrarse en el cliente o poder conocer la estructura de las tablas en la BD.
Ejemplo paso a paso de cómo explotar la vulnerabilidad
<p>Al introducir <a href="http://localhost:8080/show_question?id=2%20AND%20loquesea=4">http://localhost:8080/show_question?id=2%20AND%20loquesea=4</a> se genera la consulta: <code>SELECT author,title,time,tags,body FROM Questions WHERE id=2 AND loquesea=4</code> y la app devuelve un error de que el campo <code>loquesea</code> no existe:</p> <hr/> <h2>Error: 500 Internal Server Error</h2> <p>Sorry, the requested URL 'http://localhost:8080/show_question?id=2%20AND%20loquesea=4' caused an error:</p> <pre>Internal Server Error</pre> <h3>Exception:</h3> <pre>OperationalError('no such column: loquesea',)</pre> <h3>Traceback:</h3> <pre>Traceback (most recent call last):   File "/usr/local/lib/python3.5/site-packages/bottle.py", line 862, in _handle     return route.call(**args)   File "/usr/local/lib/python3.5/site-packages/bottle.py", line 1732, in wrapper     rv = callback(*a, **ka)   File "/Users/pep/dev/GIW/web-sec-3/coladero.py", line 59, in show_question     cur.execute(query1) sqlite3.OperationalError: no such column: loquesea</pre> <hr/> <p>por lo que se podría ir probando hasta que dicho error no apareciese lo que indicaría que el campo sí que existe:</p>

# Error: 500 Internal Server Error

Sorry, the requested URL 'http://localhost:8080/show\_question?id=2%20AND%20time=4' caused an error:

Internal Server Error

## Exception:

```
TypeError("'NoneType' object is not subscriptable",)
```

## Traceback:

```
Traceback (most recent call last):
  File "/usr/local/lib/python3.5/site-packages/bottle.py", line 862, in _handle
    return route.call(**args)
  File "/usr/local/lib/python3.5/site-packages/bottle.py", line 1732, in wrapper
    rv = callback(*a, **ka)
  File "/Users/pep/dev/GIW/web-sec-3/coladero.py", line 64, in show_question
    return template("message_detail.html", q=question, replies=replies, ident=ident)
  File "/usr/local/lib/python3.5/site-packages/bottle.py", line 3609, in template
    return TEMPLATES[tplid].render(kwargs)
  File "/usr/local/lib/python3.5/site-packages/bottle.py", line 3399, in render
    self.execute(stdout, env)
  File "/usr/local/lib/python3.5/site-packages/bottle.py", line 3386, in execute
    eval(self.co, env)
  File "/Users/pep/dev/GIW/web-sec-3/views/message_detail.html", line 45, in <module>
    <td class="title">{{!q[1]}}</td>
TypeError: 'NoneType' object is not subscriptable
```

## Medidas para mitigar la vulnerabilidad

1. No mostrar información de errores en entornos de producción. Desactivar `debug=True`
2. Validar que el id pasado sea realmente un número entero con `isinstance(id, int)`
3. Sanitizar quitando comillas, y otros símbolos.