

INFORME DE VULNERABILIDAD

Ruta(s) de la aplicación involucrada(s)

http://localhost:8080/show_all_questions
http://localhost:8080/search_question?tag=param

Tipo de vulnerabilidad

Reflected XSS

Causante de la vulnerabilidad

1. El programador de la vista:

- Plantilla: messages_search.html
- Error cometido: El carácter especial “!” indica que no se escapa el contenido de la variable. De esta forma cuando el código es ejecutado en el nivel de presentación, el contenido es interpretado por el navegador como si fuera más código de javascript en lugar de tratarlo como texto.

```
<h2>Resultados para la etiqueta: '{{!tag}}'</h2>
```

2. El programador del servidor:

- Función: search_question
- Error cometido: Aunque el programador realiza una consulta mediante el comando execute que escapa los caracteres que podrían dar lugar a ataques de tipo inyección SQL este no ha hecho nada para evitar que la variable que va a la vista para ser mostrada pueda ejecutar código.

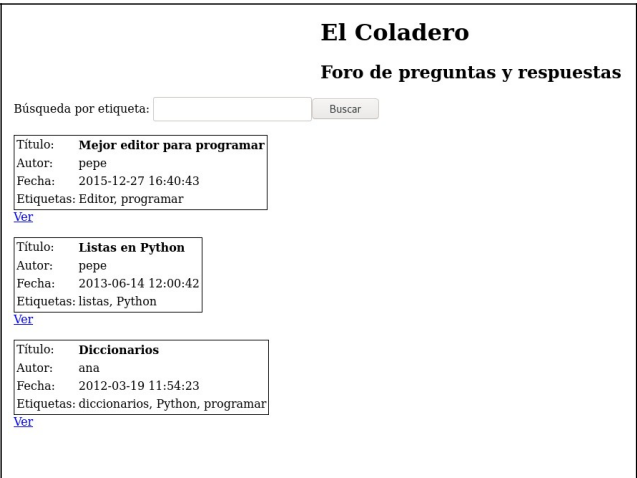



```
@get('/search_question')
def search_question():
    tag = request.query['tag']
    conn = sqlite3.connect(DBPATH)
    cur = conn.cursor()
    qbody = """SELECT id,author,title,time,tags
                FROM Questions
                WHERE tags LIKE :pattern
                ORDER BY time DESC"""
    params = {'pattern': '%' + tag + '%'}
    cur.execute(qbody,params)
    res = list(cur.fetchall())
    conn.close()
    return template('messages_search.html', questions=res, tag=tag)
```

Situaciones peligrosas o no deseadas que puede provocar

1. Mostrar las cookies que hay almacenadas en nuestro lado cliente, de esta forma podemos intentar averiguar quienes están haciendo un seguimiento de nuestras acciones y que datos les resultan de interés en lo que respecta a nosotros como para querer almacenarlos.

Ejemplo paso a paso de cómo explotar la vulnerabilidad

- Ejemplo para obtener las cookies del navegador y los datos que almacenan:

1. Ir al menu principal introduciendo la ruta <code>http://localhost:8080/show_all_questions</code>	
2. Una vez allí nos centraremos en el buscador.	
3. Para obtener las cookies y los datos solamente es necesario insertar este script <code><script> document.write(document.cookie) </script></code>	
4. Una vez insertado el script se nos redirige a <code>http://localhost:8080/search_question?tag=smtg</code> donde se puede observar un resultado similar.	

Medidas para mitigar la vulnerabilidad

1. El programador de la vista ha de escapar los caracteres de las variables a mostrar en la plantilla para tratar su contenido como si fuera texto y no permitir de esta forma que ningún script u otros códigos modifiquen la página o el funcionamiento de la aplicación. Para lograr esto basta con quitar el carácter especial “!” que precede a la variable.
2. El programador del servidor deberá implementar los mecanismos necesarios para validar la entrada del usuario de forma que no le de la posibilidad de ejecutar etiquetas web. Para solventar esto podríamos hacer un reemplazo en las variables de los caracteres `<` y `>` a sus equivalentes HTML `<` y `>`, de esta forma el navegador no interpretaría las etiquetas ya que se le estaría indicando que esos caracteres son texto plano y no el comienzo o final de una etiqueta.

INFORME DE VULNERABILIDAD

Ruta(s) de la aplicación involucrada(s)

http://localhost:8080/show_question?id=param

Tipo de vulnerabilidad

Stored XSS

Causante de la vulnerabilidad

1. El programador de la vista:

- Plantilla: message_detail.html
- Error cometido: El carácter especial “!” indica que no se escapa el contenido de la variable. De esta forma cuando el código es ejecutado en el nivel de presentación, el contenido es interpretado por el navegador como si fuera más código de javascript en lugar de tratarlo como texto.

```
% for r in replies:
<table class="reply">
  <tr>
    <td>Autor:</td>
    <td>{{!r[0]}}</td>
  <tr>
    <td>Fecha:</td>
    <td>{{!r[1]}}</td>
  <tr>
    <td>Cuerpo:</td>
    <td>{{!r[2]}}</td>
  <tr>
</table>
</br>
% end
```

2. El programador del servidor:

- Función: insert_reply
- Error cometido: Aunque el programador utiliza la inserción mediante el comando execute que escapa los caracteres que podrían dar lugar a ataques de tipo inyección SQL, no ha hecho nada para escapar los caracteres problemáticos que podrían dar lugar a la inyección de scripts, como por ejemplo: < , > , etc.

```
@post('/insert_reply')
def insert_reply():
    author = request.forms['author']
    body = request.forms['body']
    question_id = request.forms['question_id']
    conn = sqlite3.connect(DBPATH)
    cur = conn.cursor()
    qbody = """INSERT INTO Replies(author,body,time,question_id)
              VALUES (:author, :body, CURRENT_TIMESTAMP, :question_id)"""
    params = {'author': author, 'body': body, 'question_id': question_id}
    cur.execute(qbody, params)
    conn.commit()
    conn.close()
    return "Contestación insertada con éxito"
```

Situaciones peligrosas o no deseadas que puede provocar

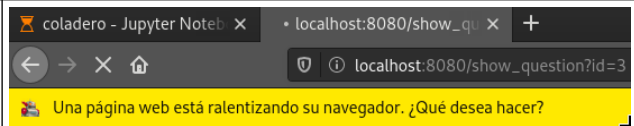
1. Puede paralizar el servidor web, por otro lado produce que una pestaña del navegador del lado del cliente se quede atascada intentando obtener un recurso que no le va a ser proporcionado nunca.

Ejemplo paso a paso de cómo explotar la vulnerabilidad

- Ejemplo para paralizar servidor y mantener en una continua espera activa a una pestaña del navegador en el lado del cliente:

1. Ir al menu principal introduciendo la ruta <code>http://localhost:8080/show_all_questions</code>	<div>El Coladero Foro de preguntas y respuestas Búsqueda por etiqueta: <input type="text"/> <input type="button" value="Buscar"/> <div>Título: Mejor editor para programar Autor: pepe Fecha: 2015-12-27 16:40:43 Etiquetas: Editor, programar Ver</div><div>Título: Listas en Python Autor: pepe Fecha: 2013-06-14 12:00:42 Etiquetas: listas, Python Ver</div><div>Título: Diccionarios Autor: ana Fecha: 2012-03-19 11:54:23 Etiquetas: diccionarios, Python, programar Ver</div></div>
2. Una vez allí seleccionar la opción “Ver” de una pregunta cualquiera.	<div>Título: Mejor editor para programar Autor: pepe Fecha: 2015-12-27 16:40:43 Etiquetas: Editor, programar Ver</div>
3. Se nos proporcionará una nueva página con un formulario básico en el que introducir los datos de nuestra contestación. Este es susceptible a la vulnerabilidad XSS Stored.	<div>Autor: <input type="text"/> <div></div><div>Cuerpo: <div></div></div><div><input type="button" value="Contestar"/></div></div>
4. Para bloquear el servidor basta con insertar un pequeño script en cualquiera de los dos campos. Una vez escrito pulsaremos el botón “Contestar” y volveremos al menú principal mediante la dirección <code>http://localhost:8080/show_all_questions</code>	<div>Autor: <input type="text" value="The Shadow"/> <div><script>while(true);</script></div><div>Cuerpo: <div></div></div><div><input type="button" value="Contestar"/></div></div>
5. Una vez allí volveremos a seleccionar la opción “Ver” de la pregunta en la que hemos inyectado el script.	<div>Título: Mejor editor para programar Autor: pepe Fecha: 2015-12-27 16:40:43 Etiquetas: Editor, programar Ver</div>

6. Veremos que la pestaña del servidor (Coladero) se ha quedado completamente colgada, mientras que en la pestaña del usuario se nos notifica que hay una web que esta ralentizando el navegador.



Medidas para mitigar la vulnerabilidad

1. El programador de la vista ha de escapar los caracteres de las variables a mostrar en la plantilla para tratar su contenido como si fuera texto y no permitir de esta forma que ningún script u otros códigos modifiquen la página o el funcionamiento de la aplicación. Para lograr esto basta con quitar el carácter especial “!” que precede a la variable.
2. El programador del servidor deberá implementar los mecanismos necesarios para validar la entrada del usuario de forma que no le de la posibilidad de ejecutar etiquetas web. Para solventar esto podríamos hacer un reemplazo en las variables de los caracteres < y > a sus equivalentes HTML < y >, de esta forma el navegador no interpretaría las etiquetas ya que se le estaría indicando que esos caracteres son texto plano y no el comienzo o final de una etiqueta.

INFORME DE VULNERABILIDAD

Ruta(s) de la aplicación involucrada(s)

http://localhost:8080/show_all_questions

Tipo de vulnerabilidad

SQL Injection

Causante de la vulnerabilidad

1. El programador del servidor:

- Función: insert_question
- Error cometido:
 1. La query está mal parametrizada pues el texto que sustituye en los huecos no está escapado, motivo por el cual se puede inyectar código SQL.
 2. Utiliza la inserción mediante el comando executeScript el cual permite ejecutar más de una sentencia SQL.

```
@post('/insert_question')
def insert_question():
    author = request.forms['author']
    title = request.forms['title']
    tags = request.forms['tags']
    body = request.forms['body']

    conn = sqlite3.connect(DBPATH)
    cur = conn.cursor()
    qbody = """INSERT INTO Questions(author, title, tags, body, time)
              VALUES ('{0}','{1}','{2}','{3}',CURRENT_TIMESTAMP)"""
    query = qbody.format(author, title, tags, body)
    cur.executescript(query)
    conn.commit()
    conn.close()
    return "Pregunta insertada con exito"
```

Situaciones peligrosas o no deseadas que puede provocar

1. Da la oportunidad de obtener el nombre de las tablas y sus campos.
2. Permite inyectar comandos de eliminación de registros o de borrado de objetos.

Ejemplo paso a paso de como explotar la vulnerabilidad

- Ejemplo para extraer el nombre de las tablas:

1. Ir al menu principal introduciendo la ruta
http://localhost:8080/show_all_questions

El Coladero

Foro de preguntas y respuestas

Búsqueda por etiqueta:

Título: **Mejor editor para programar**
Autor: pepe
Fecha: 2015-12-27 16:40:43
Etiquetas: Editor, programar
[Ver](#)

Título: **Listas en Python**
Autor: pepe
Fecha: 2013-06-14 12:00:42
Etiquetas: listas, Python
[Ver](#)

Título: **Diccionarios**
Autor: ana
Fecha: 2012-03-19 11:54:23
Etiquetas: diccionarios, Python, programar
[Ver](#)

<p>2. Una vez allí, un poco más abajo de las preguntas introducidas encontraremos un formulario básico para introducir una nueva.</p>	<p>Autor: <input type="text"/></p> <p>Título: <input type="text"/></p> <p>Etiquetas: <input type="text"/></p> <p>Cuerpo: <input type="text"/></p> <p><input type="button" value="Preguntar"/></p>
<p>3. Para intentar obtener los nombres de las tablas bastaría con introducir el último valor de los dos campos de la inserción a mano y luego cerrar la instrucción actual. Después abriremos una nueva instrucción de consulta en el que usaremos * como nombre de campo y luego tendremos que ir probando posibles nombres para ver si en algún momento damos con el nombre de alguna tabla. Si el nombre de la tabla de la consulta no existe el servidor devolverá un error ya que no ha encontrado la tabla en la base de datos, si no hay error entonces esta existe. Finalmente cerramos la instrucción de consulta y añadimos un comentario de SQL -- para evitar ejecutar el código tras nuestra inyección.</p>	<p>Autor: <input type="text" value="The Shadow"/></p> <p>Título: <input type="text" value="Evil injection"/></p> <p>Etiquetas: <input type="text" value="SQL Injection"/></p> <p>Cuerpo: <input type="text" value="nothing important','nothing important'); SELECT * FROM <Name To Guess>; --"/></p> <p><input type="button" value="Preguntar"/></p>
<p>3.1. Resultado de probar a poner en la cláusula FROM el nombre de tabla “Answer”.</p>	<p>Error: 500 Internal Server Error</p> <p>Sorry, the requested URL 'http://localhost:8080/insert_question' caused an error</p> <p>Internal Server Error</p> <p>Exception:</p> <p>OperationalError('no such table: Answer')</p> <p>Traceback:</p> <pre>Traceback (most recent call last): File "/opt/anaconda3/lib/python3.7/site-packages/bottle.py", line 868, in _handle_return_route.call(**args) File "/opt/anaconda3/lib/python3.7/site-packages/bottle.py", line 1748, in wrapper rv = callback(*a, **ka) File "<ipython-input-3-466757fd7590>", line 62, in insert_question cur.executescript(query) sqlite3.OperationalError: no such table: Answer</pre>
<p>3.1. Resultado de probar a poner en la cláusula FROM el nombre de tabla “Question”.</p>	<p>Pregunta insertada con exito</p>

NOTA: Una vez obtenido el nombre de una tabla el mecanismo para hallar los nombres de sus campos es el mismo, se irá probando ha hacer consultas con distintos nombres hasta que demos con aquellos no produzcan un error en el servidor.

- Ejemplo de inyección de comandos de eliminación de registros o de borrado de objetos:

<p>1. Ir al menu principal introduciendo la ruta <code>http://localhost:8080/show_all_questions</code></p>	<div> <div>El Coladero</div> <div>Foro de preguntas y respuestas</div> <div> <div>Búsqueda por etiqueta: <input type="text"/></div> <div>Buscar</div> </div> <div> <div> <div>Título: Mejor editor para programar</div> <div>Autor: pepe</div> <div>Fecha: 2015-12-27 16:40:43</div> <div>Etiquetas: Editor, programar</div> </div> <div>Ver</div> </div> <div> <div> <div>Título: Listas en Python</div> <div>Autor: pepe</div> <div>Fecha: 2013-06-14 12:00:42</div> <div>Etiquetas: listas, Python</div> </div> <div>Ver</div> </div> <div> <div> <div>Título: Diccionarios</div> <div>Autor: ana</div> <div>Fecha: 2012-03-19 11:54:23</div> <div>Etiquetas: diccionarios, Python, programar</div> </div> <div>Ver</div> </div> </div>
<p>2. Una vez allí, un poco más abajo de las preguntas introducidas encontraremos un formulario básico para introducir una nueva.</p>	<div> <div>Autor: <input type="text"/></div> <div>Título: <input type="text"/></div> <div>Etiquetas: <input type="text"/></div> <div>Cuerpo: <div></div></div> <div>Preguntar</div> </div>
<p>3. Para borrar registros o las tablas bastaría con introducir el último valor de los dos campos de la inserción a mano y luego cerrar la instrucción actual. Después abriremos una nueva instrucción de eliminación o borrado usando el nombre de alguna tabla extraído anteriormente. Finalmente cerramos la instrucción de borrado o eliminación y añadimos un comentario de SQL <code>--</code> para evitar ejecutar el código tras nuestra inyección.</p>	<div> <div>Autor: The Shadow</div> <div>Título: Evil Injection</div> <div>Etiquetas: SQL Injection</div> <div>Cuerpo: <div> nothing important','nothing important'); DROP TABLE IF EXISTS Questions; -- </div> </div> <div>Preguntar</div> </div>
<p>4. Resultado de insertar la pregunta con las sentencias SQL inyectadas.</p>	<div>Pregunta insertada con exito</div>

5. Intentamos volver al menu principal
http://localhost:8080/show_all_questions

Error: 500 Internal Server Error

Sorry, the requested URL 'http://localhost:8080/show_all_questions' caused an error:

Internal Server Error

Exception:

OperationalError('no such table: Questions')

Traceback:

```
Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.7/site-packages/bottle.py", line 868, in _handle
    return route.call(**args)
  File "/opt/anaconda3/lib/python3.7/site-packages/bottle.py", line 1748, in wrapper
    rv = callback(*a, **ka)
  File "<ipython-input-3-466757fd7590>", line 44, in show_all_questions
    cur.execute(query)
sqlite3.OperationalError: no such table: Questions
```

Medidas para mitigar la vulnerabilidad

1. Parametrizar correctamente la query, esto se puede conseguir automáticamente pasando los parámetros a través de la función execute. De esta forma la propia función se ocuparía de escapar los caracteres peligrosos y así los ataques por inyección no tendrían efecto.
2. Realizar la inserción mediante el comando execute ya que este no permite ejecutar más de una sentencia SQL.