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 deshonesta 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

Inclusión de etiquetas HTML, con atributos "src=" a páginas maliciosas en las que descargar archivos infectados.

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.

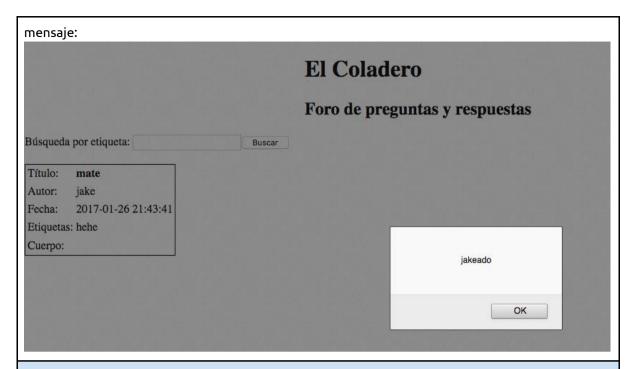
En El Coladero no hay sesiones ni autenticación del usuario, pero desde javascript se podría capturar las cookies e enviarlas a un servidor malicioso, realizando así un secuestro de sesión (session hijacking).

Ejemplo paso a paso de cómo explotar la vulnerabilidad

Se introducen los datos de la pregunta y en uno de los campos se añade el script que se va a quardar en la base de datos.

Autor:			
Título:			
Etiquet	as:	hehe	
	<	script>alert("jak	eado")
Cuerpo	: _		
Preguntar			

Al cargar el cuerpo de dicha pregunta el script se ejecuta automáticamente, mostrando el



Medidas para mitigar la vulnerabilidad

- 1. Se debería comprobar la longitud y caracteres de los campos, por ejemplo:
 - a. autor: sólo los caracteres [a-z][A-Z][0-9], '-', '_' , con longitud mínima de 3 y longitud máxima de 10
 - b. título: igual que autor pero con long. máxima de 20
 - c. etiqueta: sólo los caracteres [a-z][A-Z][0-9] y long. máxima de 200
 - d. cuerpo: longitud mínima de 5 y máxima de 2000
- 2. 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

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.

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.

Ejemplo paso a paso de cómo explotar la vulnerabilidad

Búsqueda por etiqueta: <script>alert("hello")</scrip Buscar

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

http://localhost:8080/search_question?tag=%3Cscript%3Ealert%28%22hello%22 %29%3C%2Fscript%3E_a otra persona, esta sería víctima del ataque.

Medidas para mitigar la vulnerabilidad

1. Permitir sólo los caracteres posibles de las etiquetas, es decir, [a-z][A-Z][0-9] y una longitud máxima de 20.

Informe de vulnerabilidad

Ruta(s) de la aplicación involucrada(s)

POST /insert_question

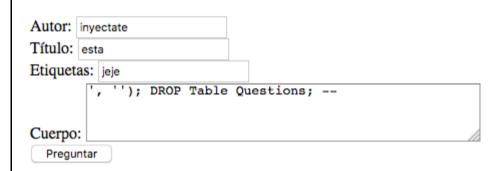
Tipo de vulnerabilidad

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



Al usarse la función executescript de sqlite3 se permite la ejecución de varias sentencias SQL a la vez, por lo que introduciendo lo anterior, se consigue terminar la consulta legítima con ', ''); y posteriormente se procede a ejecutar la sentencia deseada sobre la base de datos, por último se comenta con -- lo último de la consulta legítima.

Quedando así la consulta completa:

Medidas para mitigar la vulnerabilidad

- 1. Sólo se debería permitir la ejecución de una sentencia SQL usando la función execute().
- 2. Se deberían escapar los caracteres, sobre todo las comillas simples, usando los argumentos de la función execute, ya que esta se encarga de sanatizar los datos ante posibles inyecciones SQL.
- 3. Se debería crear un nuevo usuario en la base de datos con permisos exclusivamente para insertar en la tabla Questions y no para borrar filas o tablas enteras.

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

Al introducir http://localhost:8080/show_question?id=2%20AND%20loquesea=4 se genera la consulta: SELECT author, title, time, tags, body FROM Questions WHERE id=2 AND loquesea=4 y la app devuelve un error de que el campo loquesea no existe:

Error: 500 Internal Server Error

Sorry, the requested URL 'http://localhost:8080/show_question?id=2%20AND%20loquesea=4' caused an error:

Internal Server Error

Exception:

```
OperationalError('no such column: loquesea',)
```

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 59, in show_question
    cur.execute(query1)
sqlite3.OperationalError: no such column: loquesea
```

por lo que se podría ir probando hasta que dicho error no apareciese lo que indicaría que el campo sí que existe:

Error: 500 Internal Server Error

 $Sorry, the \ requested \ URL \ \verb|'http://localhost:8080/show_question?id=2\$20AND\$20time=4' \ caused \ an \ error: \ and \ an$

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>
        {{!q[1]}}

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.