Práctica 11 – Auditoría de seguridad

Fernando González Zamorano, Jorge Borja García, Jin Wang Xu y Gonzalo Figueroa del Val 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.

INFORME DE VULNERABILIDAD

Ruta(s) de la aplicación involucrada(s)

GET /show_all_questions y POST /insert_question

Tipo de vulnerabilidad

SQL Injection

Causante de la vulnerabilidad

La función de insertar una pregunta es el origen de esta vulnerabilidad, en concreto, las siguientes líneas de código (pertenecientes a la función insert_question()) en las que encontramos la función executescript() la cual permite ejecutar más de una consulta SQL.

```
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)
```

Situaciones peligrosas o no deseadas que puede provocar

El atacante puede realizar consultas inapropiadas al servidor, como por ejemplo comprobar el nombre de alguna tabla para posteriormente extraer información, modificar la misma, o borrar los datos de la BD.

Ejemplo paso a paso de cómo explotar la vulnerabilidad

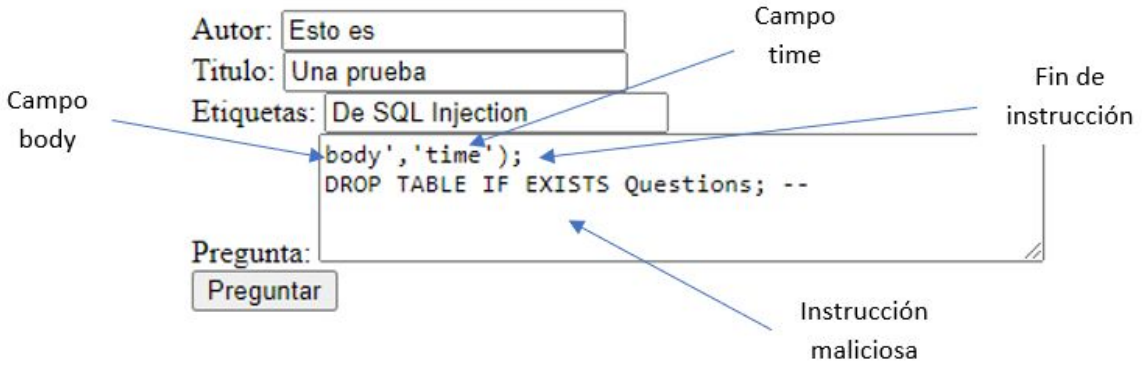
1. En el página principal (ruta http://127.0.0.1:5000/show_all_questions)

2. Debajo de todas las preguntas, encontramos un apartado para añadir una, aquí es donde podremos vulnerar la aplicación, concretamente utilizando el campo “body”. Queremos borrar los registros de una tabla, para ello, necesitamos saber el nombre de la misma. Este nombre, si no lo conocemos, se puede adivinar (en este caso imaginamos que para las preguntas existirá una tabla “Preguntas” o “Questions” o un nombre similar). Podemos intentar adquirir el nombre de la tabla primero para luego borrar, o directamente probar nombres utilizando el comando para borrar tablas (nosotros realizaremos esta segunda opción).

3. Rellenamos los campos de autor, título y etiquetas con la información que queramos, donde insertaremos el comando SQL, será en el campo “body”. Aquí, introducimos los dos últimos campos de la inserción a mano (que serán body y time, según el código).

```
qbody = """INSERT INTO Questions(author, title, tags, body, time)
          VALUES ('{0}','{1}','{2}','{3}',CURRENT_TIMESTAMP)"""
```

4. Seguidamente, cerramos la instrucción SQL y abrimos una nueva instrucción con la función que queramos ejecutar, en este caso un “DROP TABLE”. Finalmente cerramos la instrucción y añadimos un comentario de SQL (“--”) para evitar ejecutar el código tras nuestra inyección.



5. Tras pulsar en “Preguntar”, el servidor nos devolverá un error en caso de que la tabla no exista (“no such table: Questions”) o actuará como si todo hubiese funcionado correctamente llevándonos a la pantalla “insert_question” donde aparece el mensaje “Pregunta insertada con éxito” (esto es lo que nos ocurrirá a nosotros al insertar una tabla existente). Cuando regresemos a la pantalla principal (“show_all_question”), nos aparecerán errores advirtiendo de que no existe la tabla Questions (esto ocurre porque la hemos eliminado).

sqlite3.OperationalError

```
sqlite3.OperationalError: no such table: Questions
```

Traceback (most recent call last)

File "C:\Users\gonza\anaconda3\Lib\site-packages\flask\app.py", line *2464*, in __call__

```
return self.wsgi_app(environ, start_response)
```

File "C:\Users\gonza\anaconda3\Lib\site-packages\flask\app.py", line *2450*, in wsgi_app

```
response = self.handle_exception(e)
```

File "C:\Users\gonza\anaconda3\Lib\site-packages\flask\app.py", line *1867*, in handle_exception

```
reraise(exc_type, exc_value, tb)
```

File "C:\Users\gonza\anaconda3\Lib\site-packages\flask_compat.py", line *39*, in reraise

```
raise value
```

File "C:\Users\gonza\anaconda3\Lib\site-packages\flask\app.py", line *2447*, in wsgi_app

```
response = self.full_dispatch_request()
```

File "C:\Users\gonza\anaconda3\Lib\site-packages\flask\app.py", line *1952*, in full_dispatch_request

```
rv = self.handle_user_exception(e)
```

File "C:\Users\gonza\anaconda3\Lib\site-packages\flask\app.py", line *1821*, in handle_user_exception

```
reraise(exc_type, exc_value, tb)
```

File "C:\Users\gonza\anaconda3\Lib\site-packages\flask_compat.py", line *39*, in reraise

```
raise value
```

File "C:\Users\gonza\anaconda3\Lib\site-packages\flask\app.py", line *1950*, in full_dispatch_request

```
rv = self.dispatch_request()
```

File "C:\Users\gonza\anaconda3\Lib\site-packages\flask\app.py", line *1936*, in dispatch_request

```
return self.view_functions[rule.endpoint](**req.view_args)
```

File "C:\Users\gonza\Desktop\GIW\Practica 11\coladero.py", line *49*, in show_all_questions

```
cur.execute(query)
```

sqlite3.OperationalError: no such table: Questions

Medidas para mitigar la vulnerabilidad

1. Una medida para mitigar esta vulnerabilidad sería utilizar la función execute() en vez de executescript(), ya que esta última permite realizar más de una consulta SQL de una vez.
2. Otra medida para mitigarlo sería validar todos los campos que introduzca el usuario, para asegurarse que no tienen valores diferentes que puedan ser peligrosos.
3. Además, se podrían realizar consultas preconstruidas, para que al ejecutar la consulta los huecos de los campos se rellenen solo con los valores requeridos.
4. Como última medida, comprobar que la entrada no tiene comillas simples o ciertos caracteres inapropiados, y para ello, habrá que escapar el valor introducido o incluso impedir la consulta (en caso de tenerlas).

INFORME DE VULNERABILIDAD

Ruta(s) de la aplicación involucrada(s)

GET /show_all_questions
GET /search_question

Tipo de vulnerabilidad

XSS Reflejado

Causante de la vulnerabilidad

El origen de esta vulnerabilidad es la función de buscar una pregunta (search_question()) que permite la ejecución de un script, al no comprobar el campo “tag”.

Situaciones peligrosas o no deseadas que puede provocar

El atacante puede aprovechar la vulnerabilidad y realizar ataques phishing, modificando el DOM de la web para crear una imagen (por ejemplo), y hacer una redirección a otra página web peligrosa, enviándole cookies con información personal, como por ejemplo, el nombre del usuario y la contraseña.

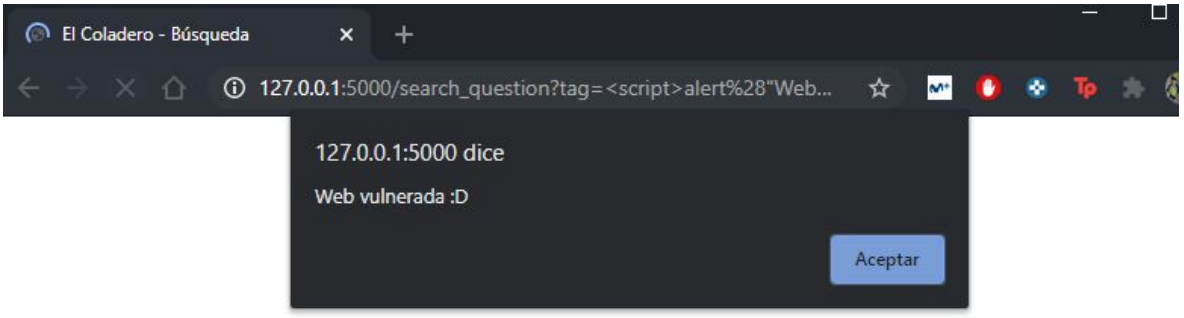
Este tipo de ataque es realmente peligroso para los usuarios ya que, si confían en la web, no serán conscientes de que su seguridad se ve afectada si interactúan con los objetos dentro de la web.

Ejemplo paso a paso de cómo explotar la vulnerabilidad

1. Tanto en la página principal como en la página “show_question” (lugares donde se encuentra la barra de búsqueda por etiqueta) podemos explotar esta vulnerabilidad.

2. En el campo de texto de la barra de búsqueda, insertaremos el script que queremos que se ejecute al pulsar en “Buscar”.

3. En este caso, probaremos con un script sencillo que hará saltar una alerta: <script>alert(“Web vulnerada :D”)</script>



4. El script que ejecutamos es sencillo y no dañino, pero las posibilidad de hacer daño con esta vulnerabilidad es muy grande (tal y como hemos explicado previamente).

Medidas para mitigar la vulnerabilidad

1. Para evitar este tipo de ataques, necesitaremos bloquear la capacidad de introducir código HTML dentro de los campos de texto. Para ello es necesario limpiar todo contenido que aparecerá dentro de la página web, realizando un escape de los caracteres sensibles como por ejemplo '<' '>'.
2. Otra medida imprescindible es validar todas las entradas de usuario para evitar que el usuario introduzca valores que no son los esperados.
3. Otra medida es comprobar todo el código HTML que va a aparecer en pantalla.

INFORME DE VULNERABILIDAD

Ruta(s) de la aplicación involucrada(s)

POST /insert_reply
GET /show_question

Tipo de vulnerabilidad

XSS Persistente

Causante de la vulnerabilidad

1. Dentro de la función insert_reply(): no realiza ninguna comprobación del contenido del campo 'body', por lo que da la posibilidad de insertar un script malicioso dentro de la BBDD que se ejecutará cada vez que se muestren las respuestas.

```
body = request.form['body']
question_id = request.form['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)
```

2. Dentro de la función show_question(): no se comprueba en ningún momento los datos obtenidos de la BBDD, por lo que se meten directamente en la variable replies que se utiliza para la mostrar la información dentro de la web.

```
cur.execute(qbody2, params)
replies = list(cur.fetchall())
conn.close()
return render_template("message_detail.html", q=question, replies=replies, ident=ident)
```

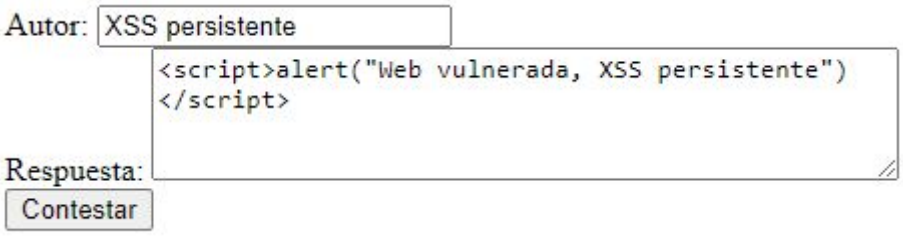
Situaciones peligrosas o no deseadas que puede provocar

Este ataque es similar al anterior (ataques phising, obtener cookies con información personal, cookies de sesión, etc.) pero aún más peligroso, ya que en lugar de afectar al usuario que está utilizando la web en ese momento, afecta a todos los que accedan al lugar de la web donde se encuentra el script insertado debido a que el código malicioso se incrusta en la BBDD y se ejecuta todas las veces que lo haga la página.

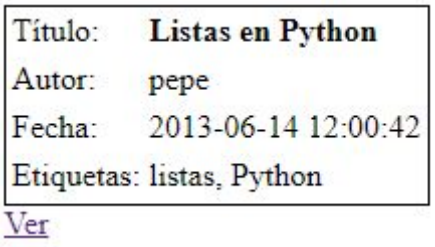
Ejemplo paso a paso de cómo explotar la vulnerabilidad

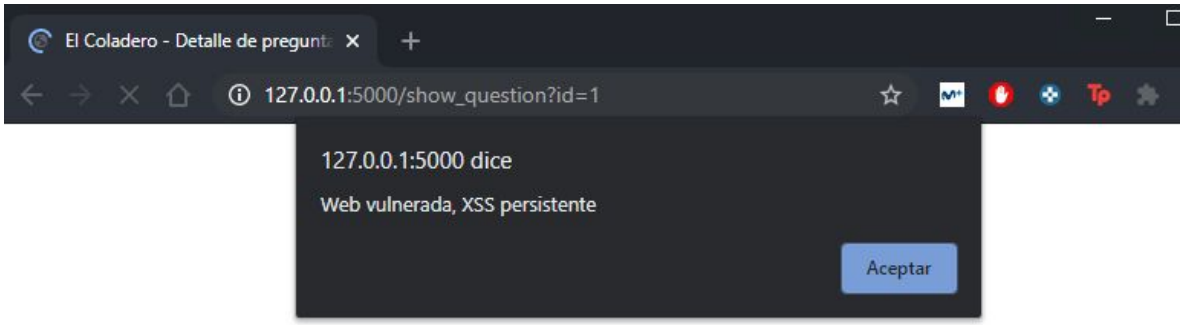
1. Para explotar esta vulnerabilidad, desde la página principal seleccionaremos una de las preguntas que aparecen (enlace “Ver”) para situarnos en la página “show_question”. Aquí, encontraremos la pregunta en concreto y las respuestas anidadas a ella. En los campos de texto que aparecen en la parte inferior de la página será donde actuemos maliciosamente, concretamente en el campo “Respuesta”.

2. En el campo “Autor” escribiremos el texto que queramos, y en el campo “Respuesta” introducimos el script que quedará almacenado en la base de datos. En este caso, insertaremos un script sencillo que hará saltar una alerta en el navegador: <script>alert(“Web vulnerada, XSS persistente”)</script>



3. Al pulsar en “Contestar”, habremos insertado el script en la base de datos, y cada vez que entremos a la página de la pregunta en cuestión (enlace “Ver”), se ejecutará el script.





4. Este script no es dañino pero muestra que la vulnerabilidad ha sido explotada. De igual manera, se pueden utilizar script maliciosos con un alto potencial de daño.

Medidas para mitigar la vulnerabilidad

1. No permitir que el usuario introduzca caracteres que no se deberían introducir en ese campo, especialmente los que tienen significado en HTML y pueden suponer un ataque como “<”, “>” y comillas simples y dobles.
2. Comprobar los valores obtenidos de la BBDD.
3. Otra medida es validar que todas las entradas de usuario son buenas y no contienen resultados erróneos o que puedan ser peligrosos.