

AUDITORÍA DE VUNERABILIDADES

Carlos Gavidia Ortiz, Iván Monterrubio Cerezo, Sergio González Jiménez y José Antonio Bernal Pérez 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 deshonesta ninguna otra actividad que pueda mejorar nuestros resultados ni perjudicar los resultados de los demás.

Inyección SQL

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

“/insert_question”

Tipo de vulnerabilidad:

Inyección SQL

Causante de la vulnerabilidad:

Executescript(), al contrario que Execute(), permite ejecutar varias sentencias SQL.

Situaciones peligrosas o no deseadas que puede provocar:

En el contenedor del cuerpo del post, cualquiera puede introducir una sentencia SQL maliciosa cuyo propósito sea alterar la base de datos. Se podrían crear, modificar o borrar tablas.

Ejemplo de cómo explotar la vulnerabilidad:

1. Ir a la ruta “/show_all_questions”. Debajo de los posts ya realizados por otros usuarios, hay un “formulario” que rellenar con nombre del autor, título, etiquetas, y el cuerpo del post. El botón “Preguntar” ejecutará el método `post Insert_question()`.

Foro de preguntas y respuestas

Búsqueda por etiqueta:

Título: **Mejor manera de 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

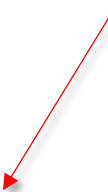
[Ver](#)

Autor:

Título:

Etiquetas:

Cuerpo:



2. Debemos escribir siempre en el *Cuerpo* “**’,CURRENT_TIMESTAMP);**” antes de escribir la/s sentencia/s que queramos inyectar. Esto se debe a que la *query* está formada, al final, por el contenido del *Cuerpo* seguido de la fecha del sistema (que se introduce desde el código). De este modo, la *query* queda completa y podemos inyectar las sentencias que queramos. En este ejemplo, borramos la tabla *Questions*.

Autor:

Título:

Etiquetas:

Cuerpo:

3. Finalmente, en este ejemplo, si vamos a la ruta “/show_all_questions”, la pantalla nos mostrará un error debido a que la tabla *Questions* ya no existe.

Error: 500 Internal Server Error

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

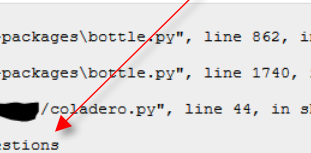
Internal Server Error

Exception:

OperationalError('no such table: Questions',)

Traceback:

```
Traceback (most recent call last):
  File "C:\Users\user\AppData\Local\Anaconda3\lib\site-packages\bottle.py", line 862, in _handle
    return route.call(**args)
  File "C:\Users\user\AppData\Local\Anaconda3\lib\site-packages\bottle.py", line 1740, in wrapper
    rv = callback(*a, **ka)
  File "C:\Users\user\AppData\Local\Anaconda3\lib\site-packages\coladero.py", line 44, in show_all_questions
    cur.execute(query)
sqlite3.OperationalError: no such table: Questions
```



Medidas para mitigar la vulnerabilidad:

Se podría emplear el método `Execute()`, del mismo modo que se usa en el método `Insert_reply()`, que requiere de una sentencia SQL preparada, la cual previene de inyecciones SQL.

XSS persistente

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

"/insert_reply" y "/show_question"

Tipo de vulnerabilidad:

XSS persistente

Causante de la vulnerabilidad:

La función insert_reply() guarda en la base de datos la respuesta que se haga a una pregunta sin validar antes la entrada del usuario y show_question() la muestra ejecutando el script que se haya guardado, en caso de que lo haya.

Situaciones peligrosas o no deseadas que puede provocar:

En el formulario de respuesta se puede introducir cualquier cadena de texto y se guardará en la base de datos si ningún tipo de procesamiento. Si el usuario introduce un script, siempre que se lea esa respuesta de la base de datos se ejecutará el script que contiene el texto de la respuesta, pudiendo relizar graves ataques como el robo de todo el DOM o un ataque phishing.

Ejemplo de cómo explotar la vulnerabilidad:

1. Ir a la ruta "/show_all_questions" y pulsar en el enlace [Ver](#) de una de las preguntas que aparecen para ver los detalles y las respuestas a esa pregunta.

Foro de preguntas y respuestas

Búsqueda por etiqueta:

Título: **Mejor manera de 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](#)

2. En el formulario que aparece en la parte inferior para guardar una respuesta, escribir el script que queremos que se guarde en la base de datos.

El Coladero

Foro de preguntas y respuestas

Búsqueda por etiqueta:

Título: **Mejor manera de programar**
Autor: pepe
Fecha: 2015-12-27 16:40:43
Etiquetas: Editor, programar
Cuerpo: Vim o Emacs?

Autor:

Cuerpo:

3. Volver a abrir el detalle de esa pregunta y comprobar que al mostrar las respuestas se ejecuta el script que hemos introducido.

[Volver al detalle de la pregunta](#)

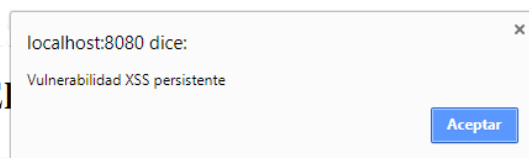
El

Foro de preguntas y respuestas

Búsqueda por etiqueta:

Título: **Mejor manera de programar**
Autor: pepe
Fecha: 2015-12-27 16:40:43
Etiquetas: Editor, programar
Cuerpo: Vim o Emacs?

Autor: Atacante
Fecha: 2018-02-05 03:19:11
Cuerpo:



Medidas para mitigar la vulnerabilidad:

Validar la entrada del usuario. Se puede comprobar que el texto no incluya etiquetas de apertura o cierre `<script>` o utilizar métodos de saneamiento de las entradas para evitar caracteres problemáticos como por ejemplo `cgi.escape()`.

XSS reflejado

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

"/show_all_questions" y "/search_question"

Tipo de vulnerabilidad:

Reflected XSS

Causante de la vulnerabilidad:

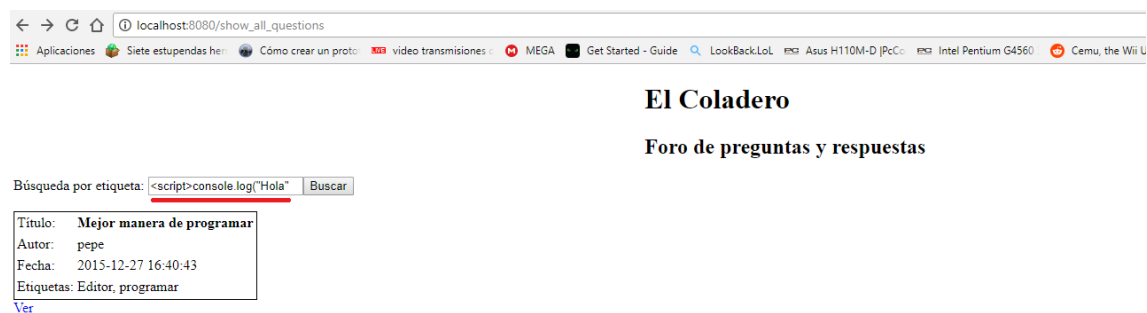
Es posible inyectar texto HTML (como un script Javascript) en el cuadro de texto en /show_all_questions para buscar preguntas.

Situaciones peligrosas o no deseadas que puede provocar:

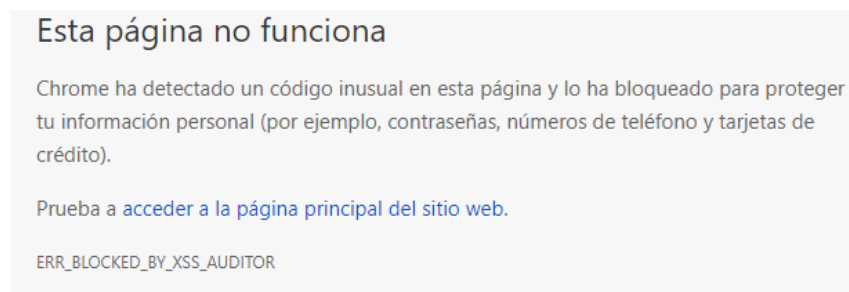
Un atacante puede utilizar esta vulnerabilidad para inyectar código Javascript en el servidor y coger cualquier tipo de dato de éste, tanto datos privados de clientes como información privilegiada.

Ejemplo de cómo explotar la vulnerabilidad:

1. Ir a la ruta /show_all_questions. En ella, hay un cuadro de texto para buscar preguntas por etiqueta.



2. Si inyectamos código Javascript en el cuadro de texto en forma de script HTML, el script se ejecutará al llegar a la ruta /search_question. Chrome muestra este error:



Medidas para mitigar la vulnerabilidad:

Es posible eliminar la vulnerabilidad escapando los caracteres HTML de la etiqueta buscada antes de añadirla a la página HTML con *html.escape(tag)*.