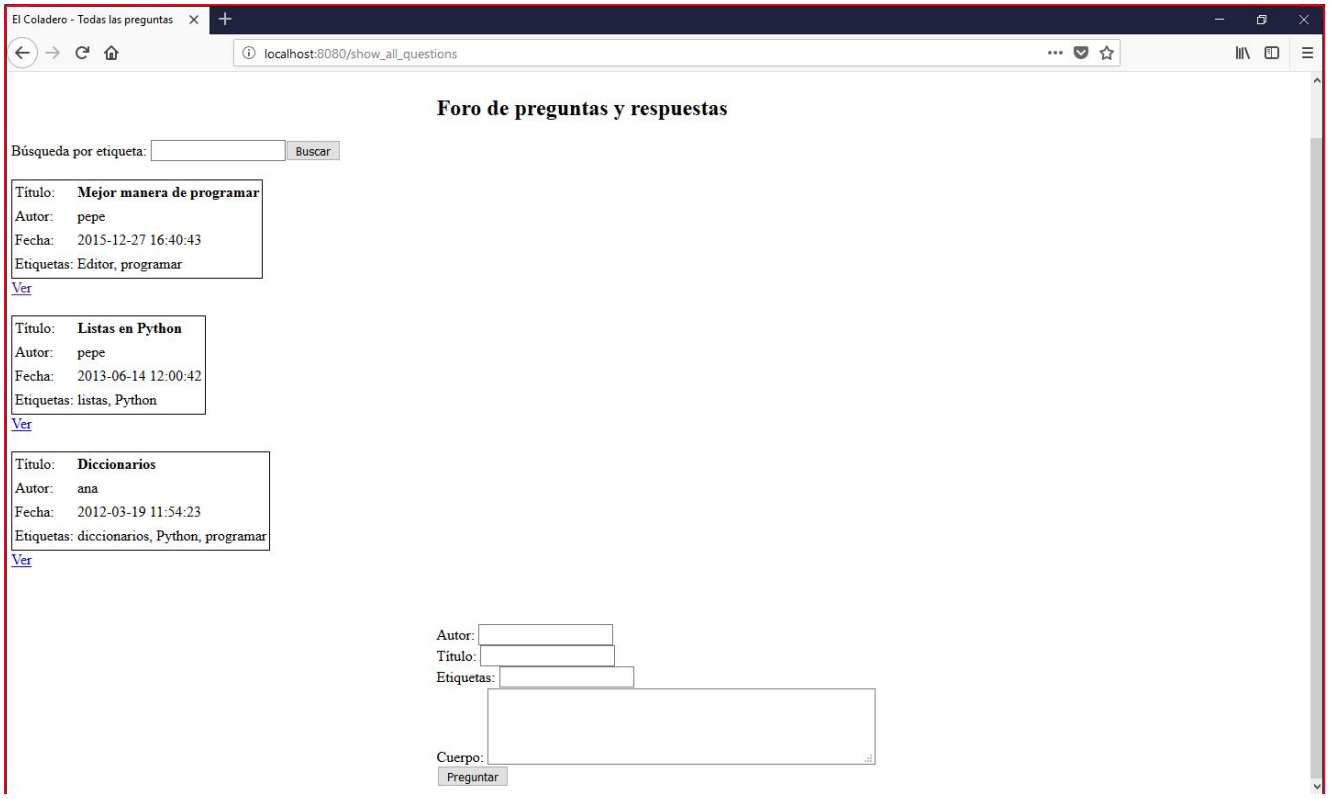
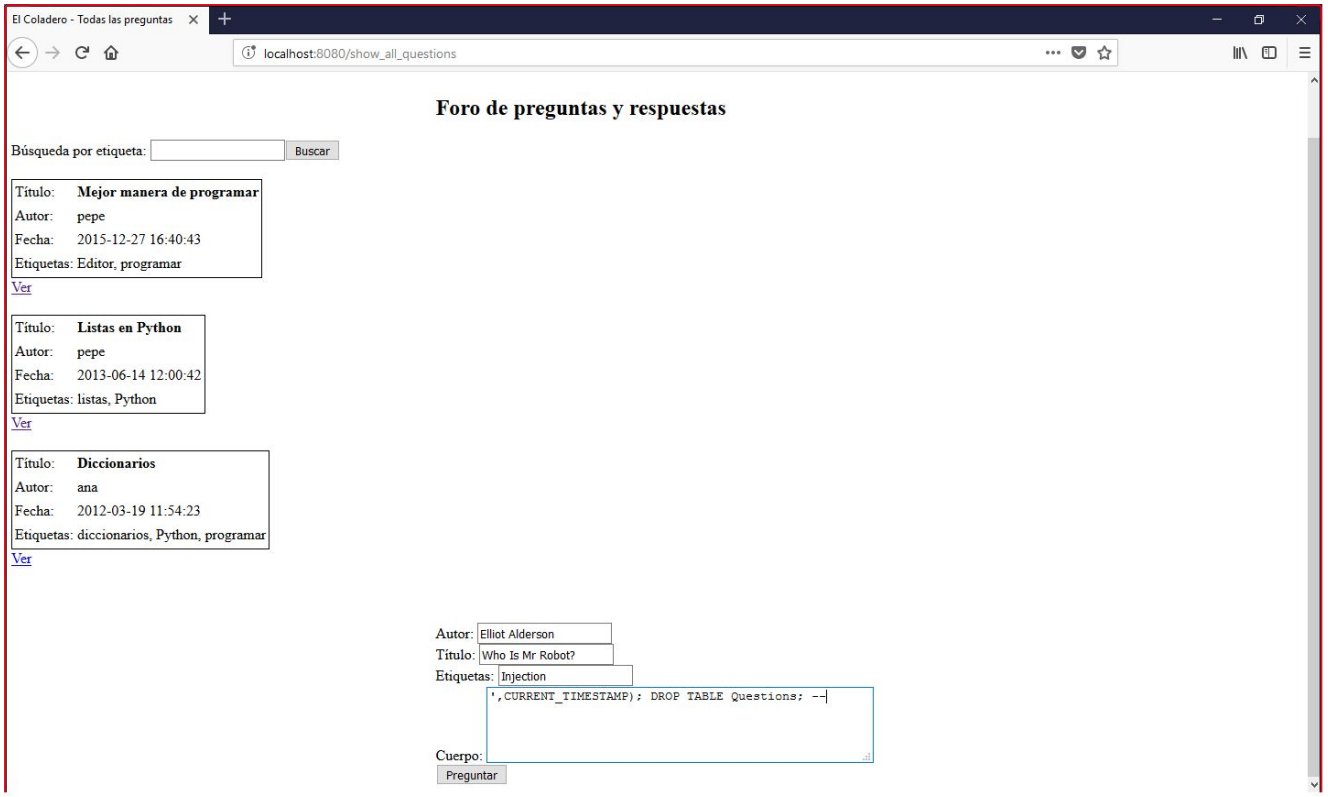
Gestión de la Información en la Web
Práctica Auditoría de Seguridad
Grupo 04
Marcos Robles Palencia y Álvaro de la Cruz Casado declaramos que esta solución es fruto exclusivamente de nuestro trabajo personal. No hemos sido ayudados por ninguna otra persona ni hemos obtenido la solución de fuentes externas, 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)
/show_all_questions e /insert_question
Tipo de vulnerabilidad
SQL Injection
Causante de la vulnerabilidad
La forma en la que se construye la consulta SQL en insert_question() qbody = """INSERT INTO Questions(author, title, tags, body, time) VALUES ('{0}','{1}','{2}','{3}',CURRENT_TIMESTAMP)""" Cómo se sustituyen los parámetros de entrada en ella (lo que hace que no se sanitice la entrada) query = qbody.format(author, title, tags, body) Y cómo se ejecuta la consulta con el cur.executescript(query), ya que permite ejecutar varias sentencias SQL.
Situaciones peligrosas o no deseadas que puede provocar
Modificación de cualquier campo de la base de datos, añadir nuevos campos o borrarlos, e incluso borrar tablas de la misma.
Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla)

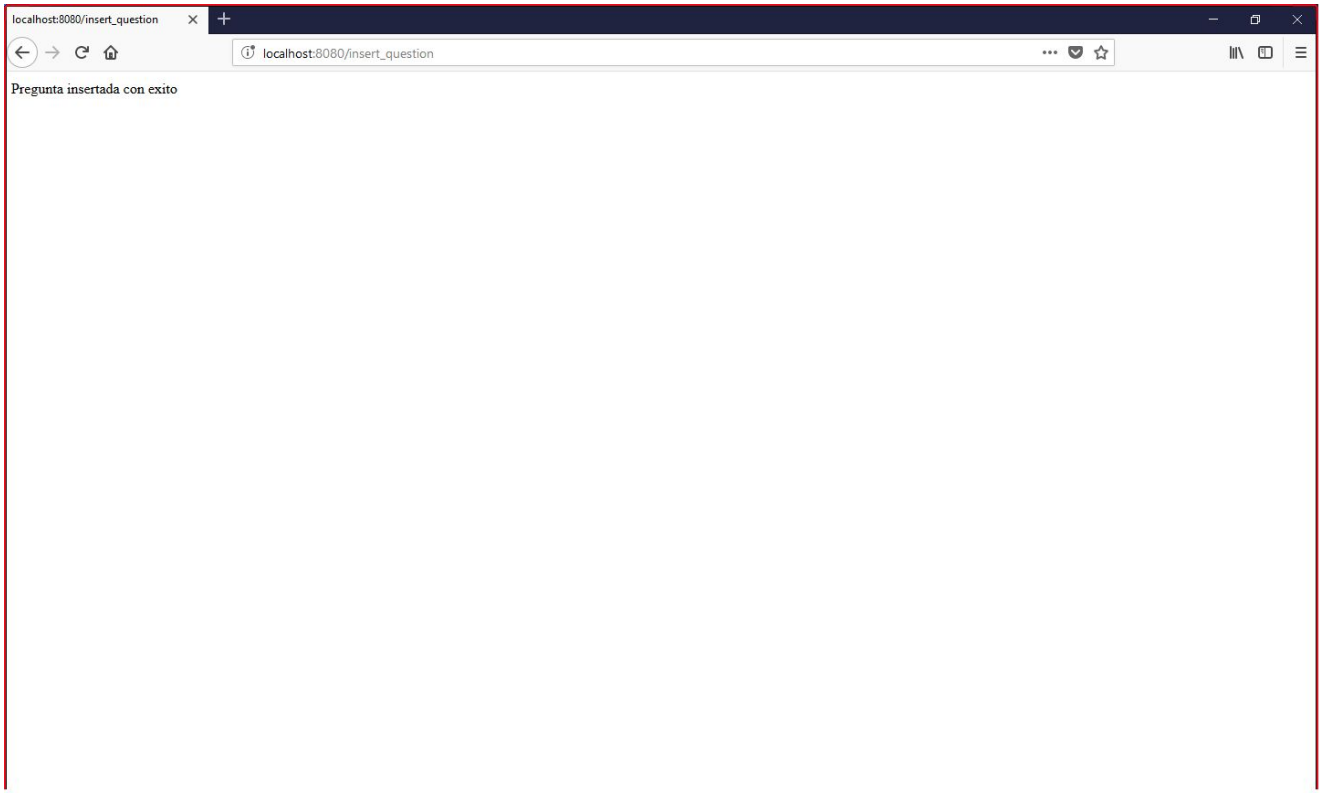


Partimos de la página principal, en la cual tenemos un formulario para introducir preguntas.

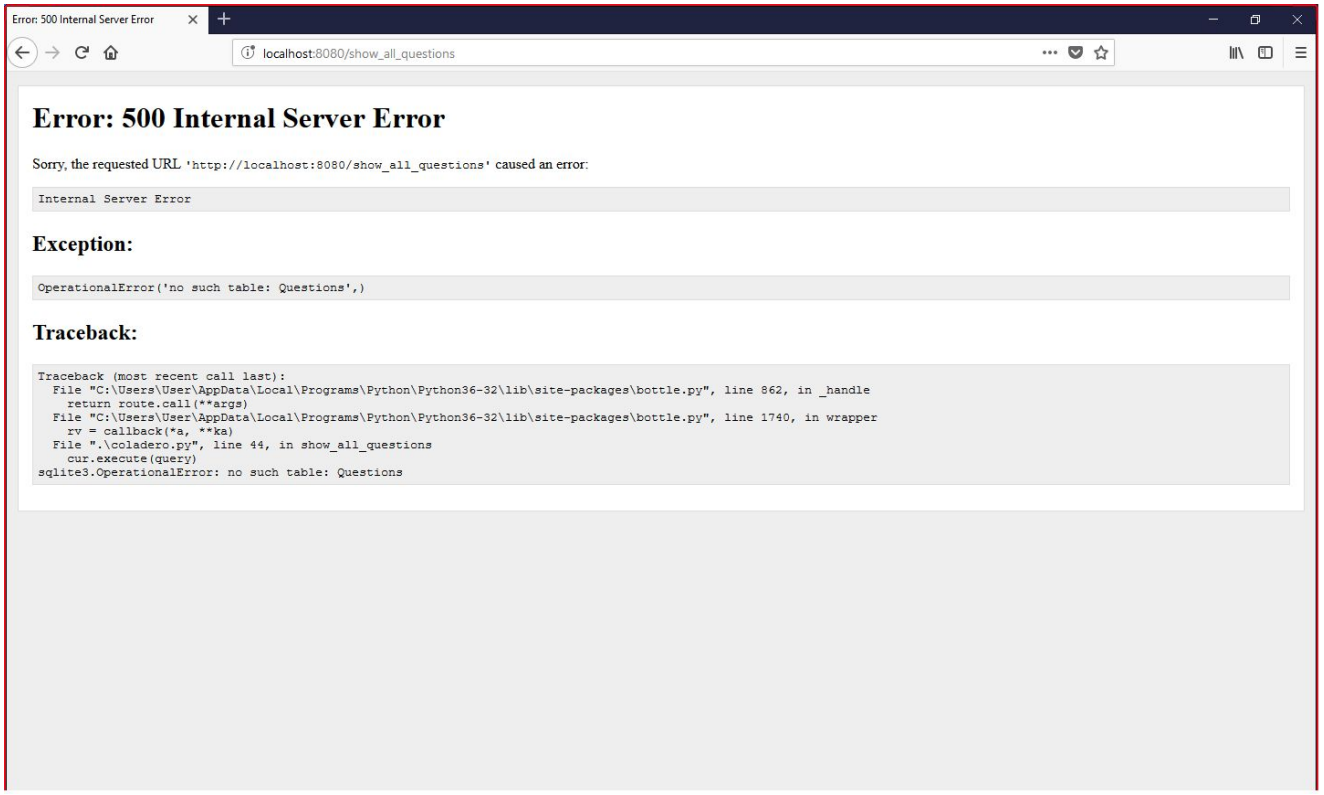


En este formulario, podemos aprovechar el campo del cuerpo del mensaje para introducir un mensaje malicioso que luego ejecutará el servidor en forma de consulta SQL si le damos la forma adecuada para conseguir ejecutarlo. Para ello debemos conocer la estructura de la consulta, ya que si no es muy complicado saber exactamente cómo crear la consulta maliciosa.

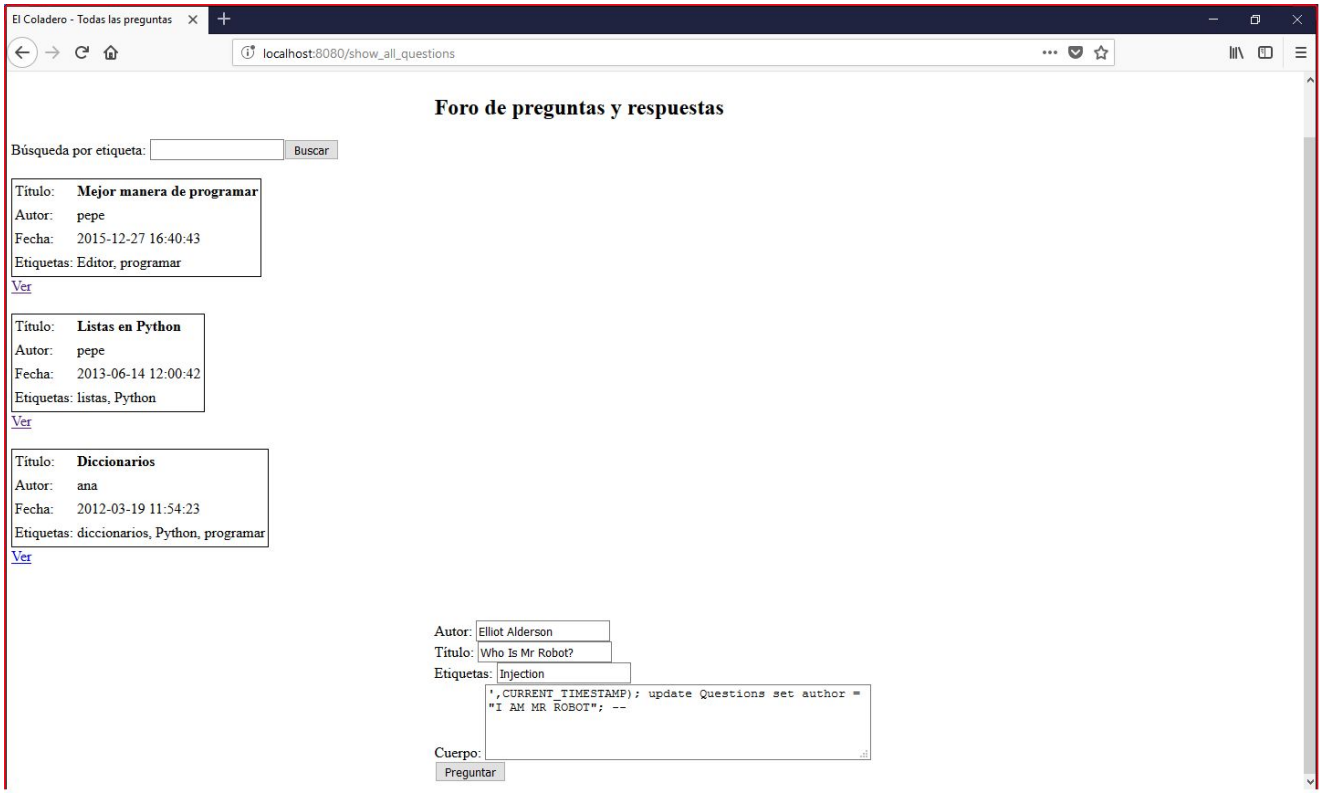
En este caso hemos introducido => ',CURRENT_TIMESTAMP); DROP TABLE Questions; --



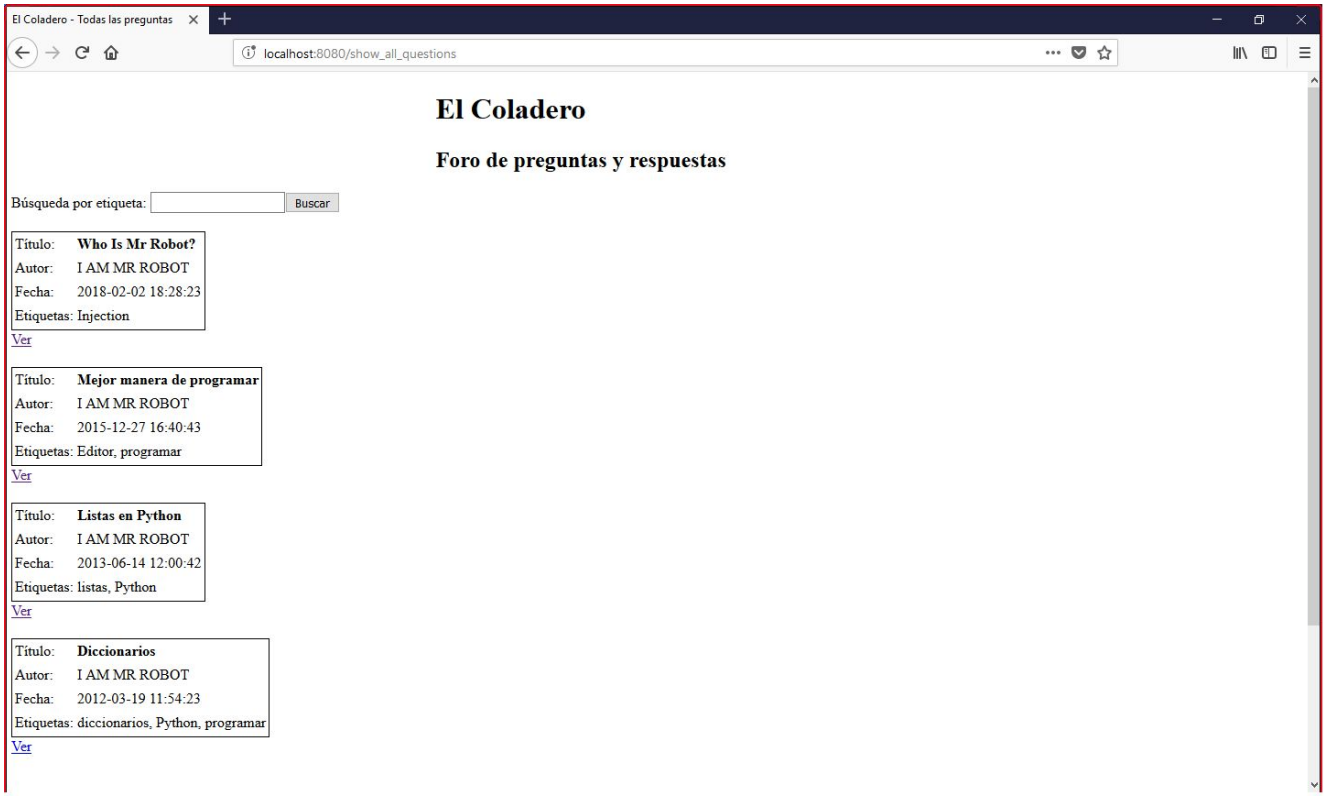
Insertamos la pregunta.



Tras intentar volver a la página de inicio, vemos un mensaje de error del servidor indicando que no existe la tabla “Questions”, ya que en nuestra consulta maliciosa hemos hecho que la borre.



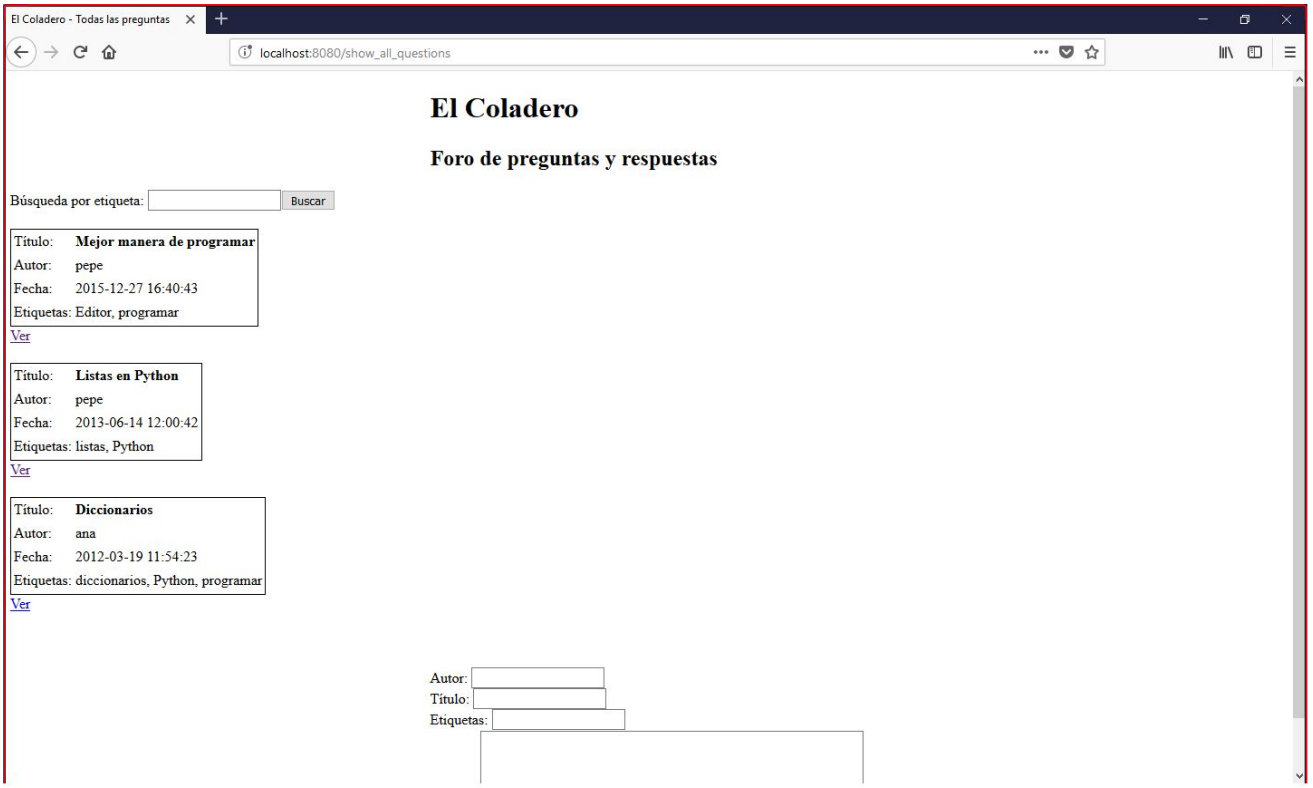
También podemos introducir una consulta que modifique campos de las tablas.



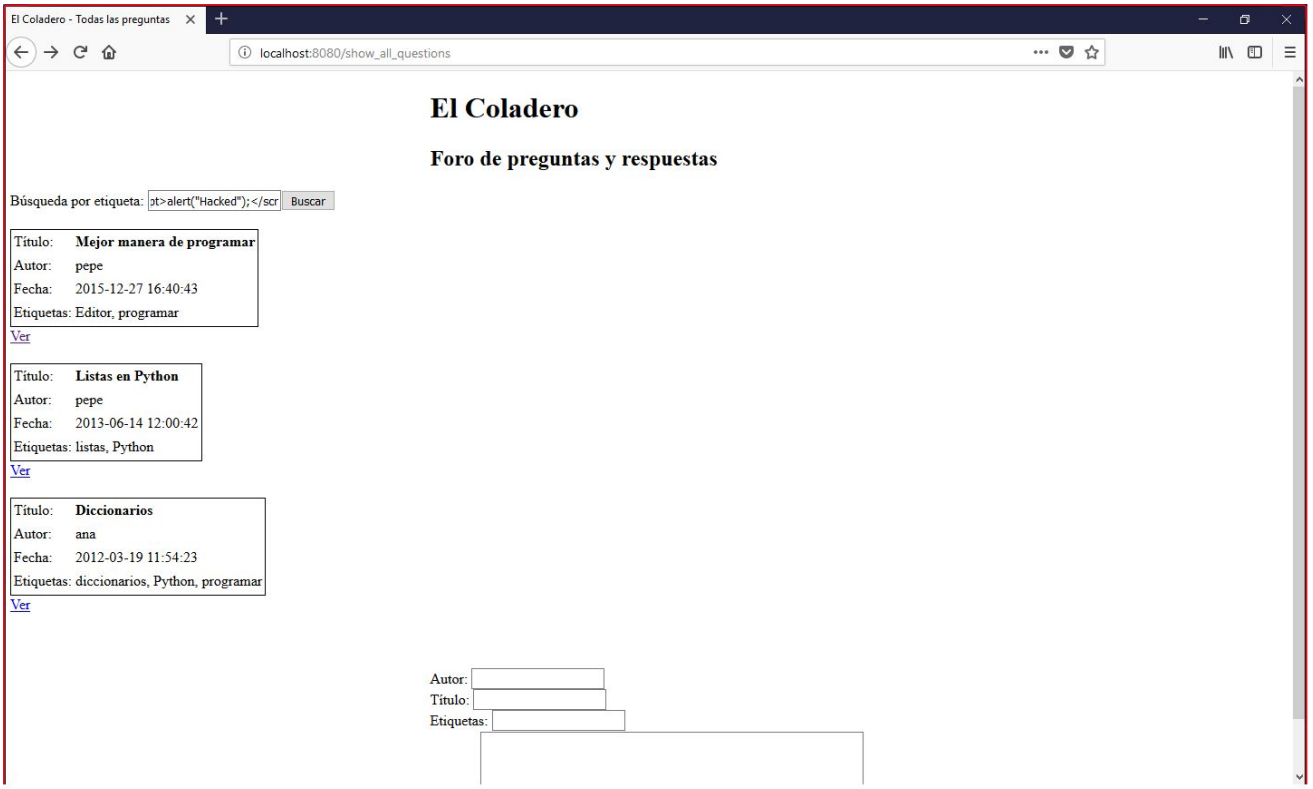
En este caso hemos modificado todos los autores.

Medidas para mitigar la vulnerabilidad
Cambiar la manera en la que formamos la consulta SQL para sanitizar la entrada. qbody = """INSERT INTO Questions(author, title, tags, body, time) VALUES ('{0}','{1}','{2}','{3}',CURRENT_TIMESTAMP)""" Pasaría a ser: qbody = """INSERT INTO Questions(author, title, tags, body, time) VALUES (:author, :title, :tags, :body, CURRENT_TIMESTAMP)""" params = {'author': author, 'title': title, 'tags': tags, 'body': body} Y en vez de cur.executescript(query), usar cur.execute(qbody, params) para que solo se permita ejecutar una sentencia SQL en vez de varias.

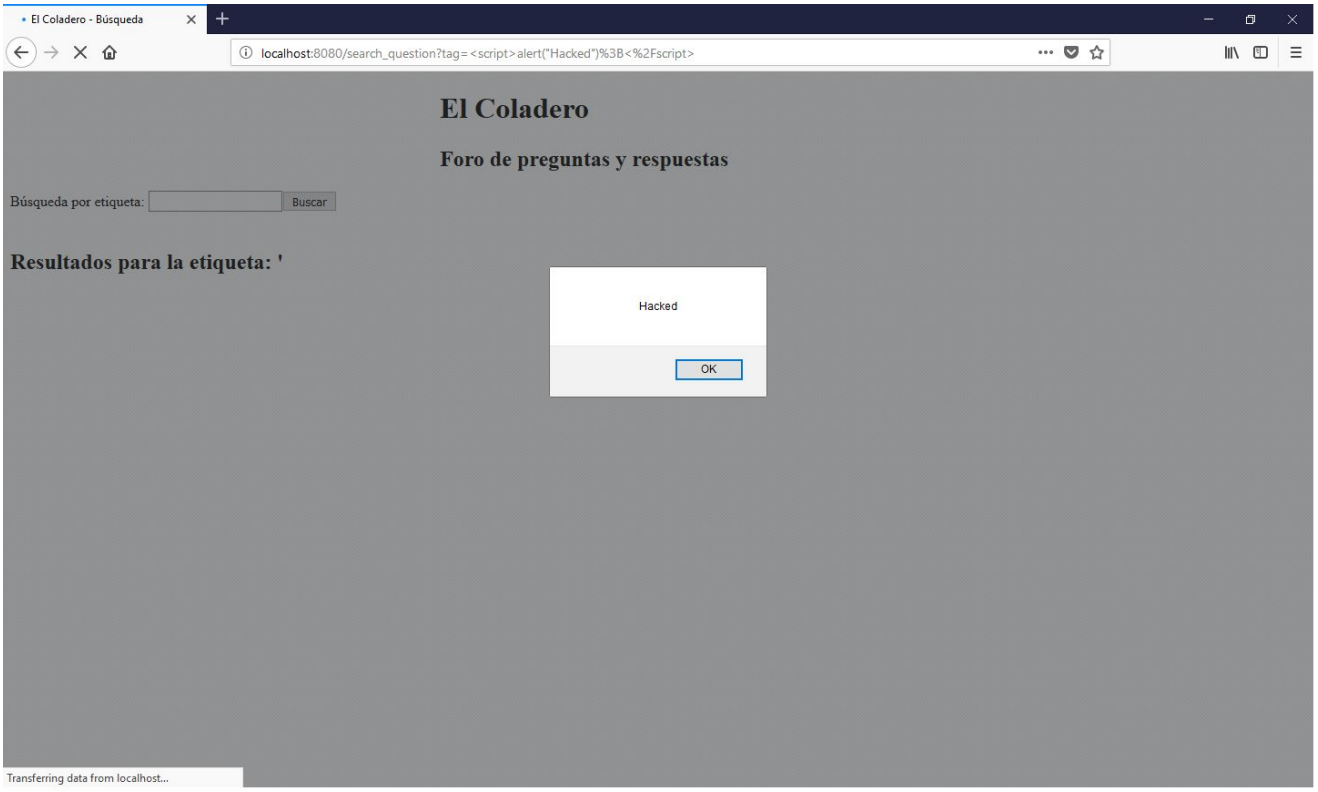
INFORME DE VULNERABILIDAD
Ruta(s) de la aplicación involucrada(s)
/show_all_questions y /search_question
Tipo de vulnerabilidad
Reflected XSS
Causante de la vulnerabilidad
La no sanitización del parámetro "tag" antes de pasarlo por parámetro al template en la función search_question(), ya que el template lo insertará tal cual en el html y por tanto si incluimos un script lo ejecutará.
Situaciones peligrosas o no deseadas que puede provocar
Robo de credenciales, de cookies, ejecución de acciones con privilegios de otro usuario, todo esto por ejemplo enviando a un usuario mediante un phising un enlace acortado (usando por ejemplo Google URL Shortener https://goo.gl/ ,para que no sospeche) a la página vulnerable incluyendo en el enlace el script con las acciones que queremos ejecutar (por eso el enlace acortado, para que de primeras no se vea y el usuario caiga). De esta forma, al hacer clic el usuario víctima sobre el enlace, ejecutará el código y de ese modo todo lo que nosotros hayamos añadido.
Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla)



Partimos de la página principal, en la cual tenemos un campo de búsqueda para filtrar los resultados por etiquetas.



Si introducimos código javascript, este se ejecutará al aplicar el filtro. En este caso hemos introducido <script>alert(“Hacked”);</script>



Al aplicar el filtro se ejecutará el código introducido, y en este caso se muestra un mensaje con el texto “Hacked”.

Medidas para mitigar la vulnerabilidad
Sanitizar la entrada del parámetro “tag” de forma que escape caracteres no deseados antes de pasarlo al template. Una opción distinta de la que aparece en las diapositivas puede ser usar la librería Bleach (pip install bleach) y hacer “tag = bleach.clean(tag)”. De esta forma al insertarlo en el html no se ejecutará el script malicioso, ya que los caracteres problemáticos como podrían ser “<” “>” se habrán sustituido para evitarlo.

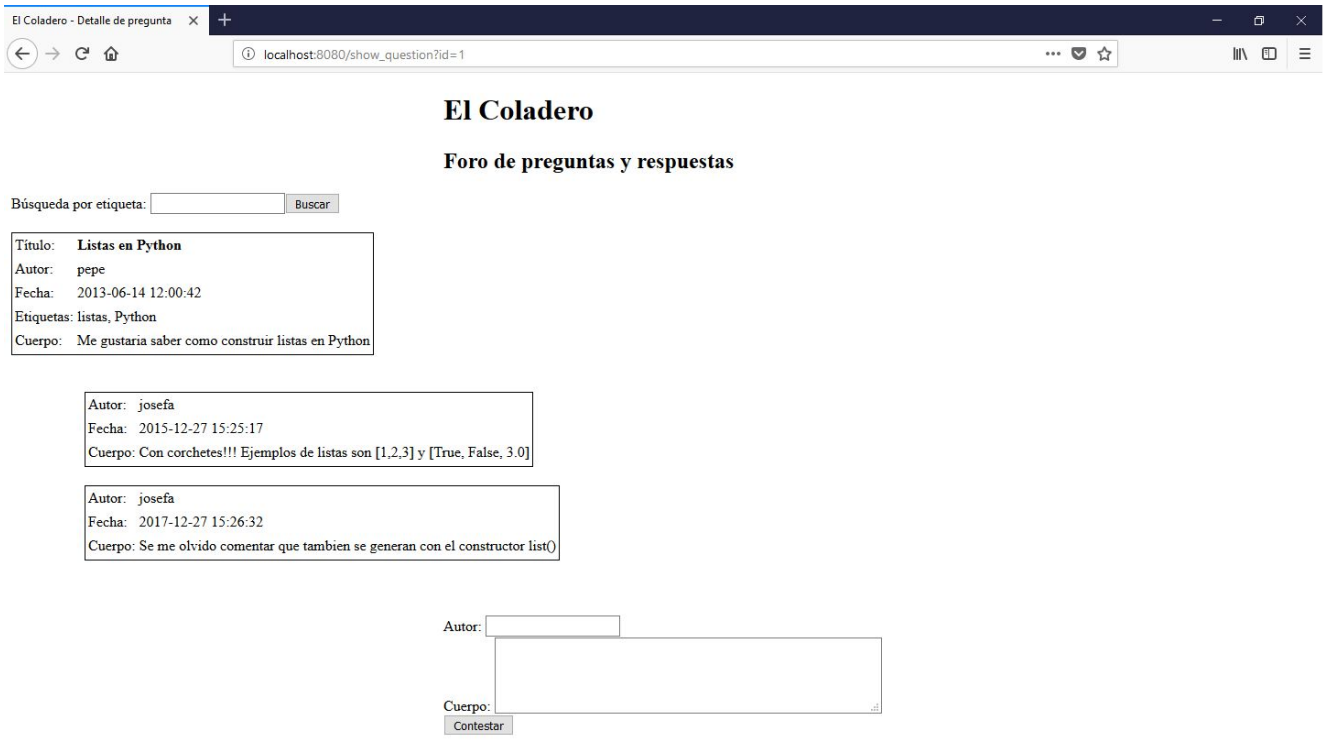
INFORME DE VULNERABILIDAD
Ruta(s) de la aplicación involucrada(s)
/show_question e /insert_reply
Tipo de vulnerabilidad
Persistent XSS
Causante de la vulnerabilidad
La no sanitización de los campos del formulario para insertar respuestas, ya que de esta forma se

insertará en texto plano lo que sea que introduzcamos, y se incrustará en el html. Así, al insertar un código javascript por ejemplo, se autoejecutará al acceder a la página.

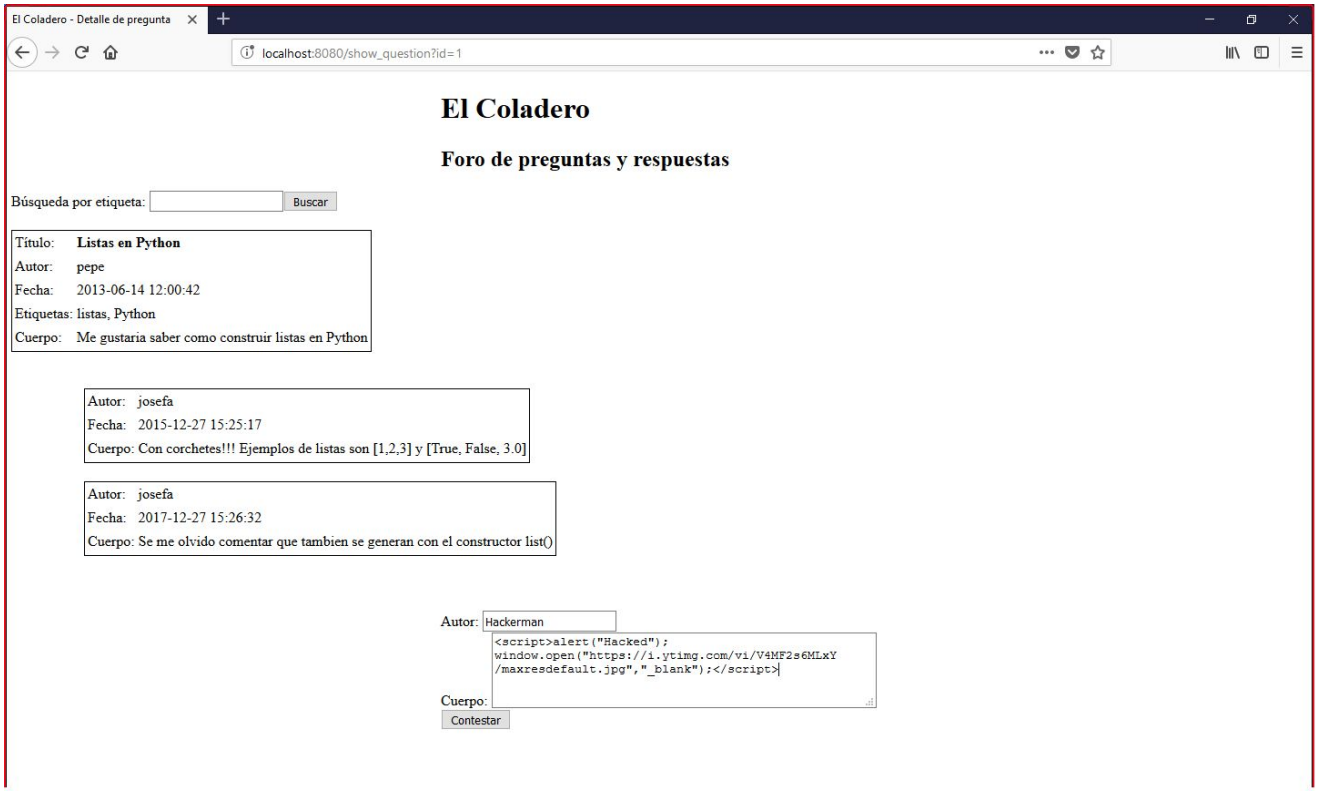
Situaciones peligrosas o no deseadas que puede provocar

Una vez se abra la página que lo aloja, se ejecutará cualquier código insertado, como robar las cookies del navegador y mandarlas al atacante, abrir páginas sin el consentimiento del usuario, robar las credenciales del usuario y mandarlas al atacante, crear páginas de phising….

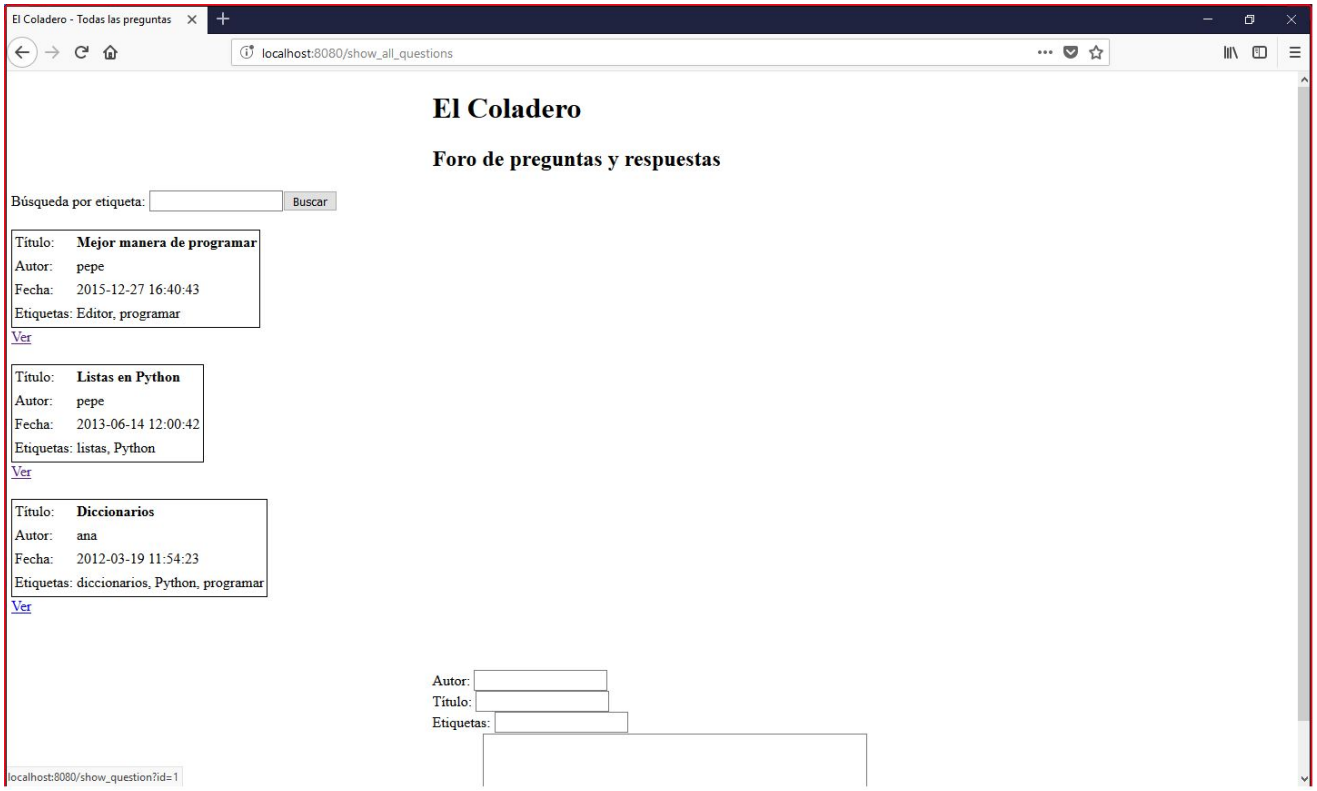
Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla)



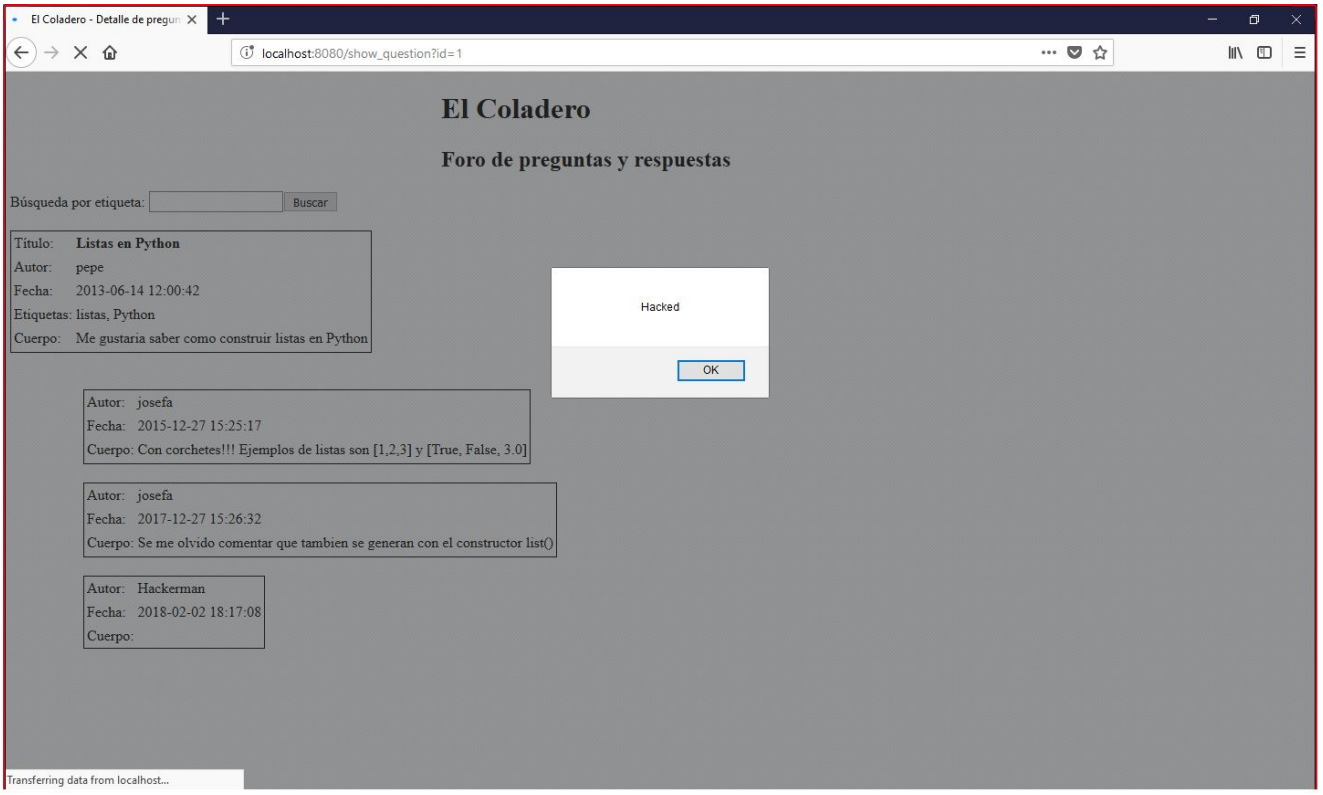
Partimos de la página de una pregunta ya formulada. En este caso la pregunta con título “Listas en Python”. En esta página disponemos de un formulario para responder a la pregunta formulada.



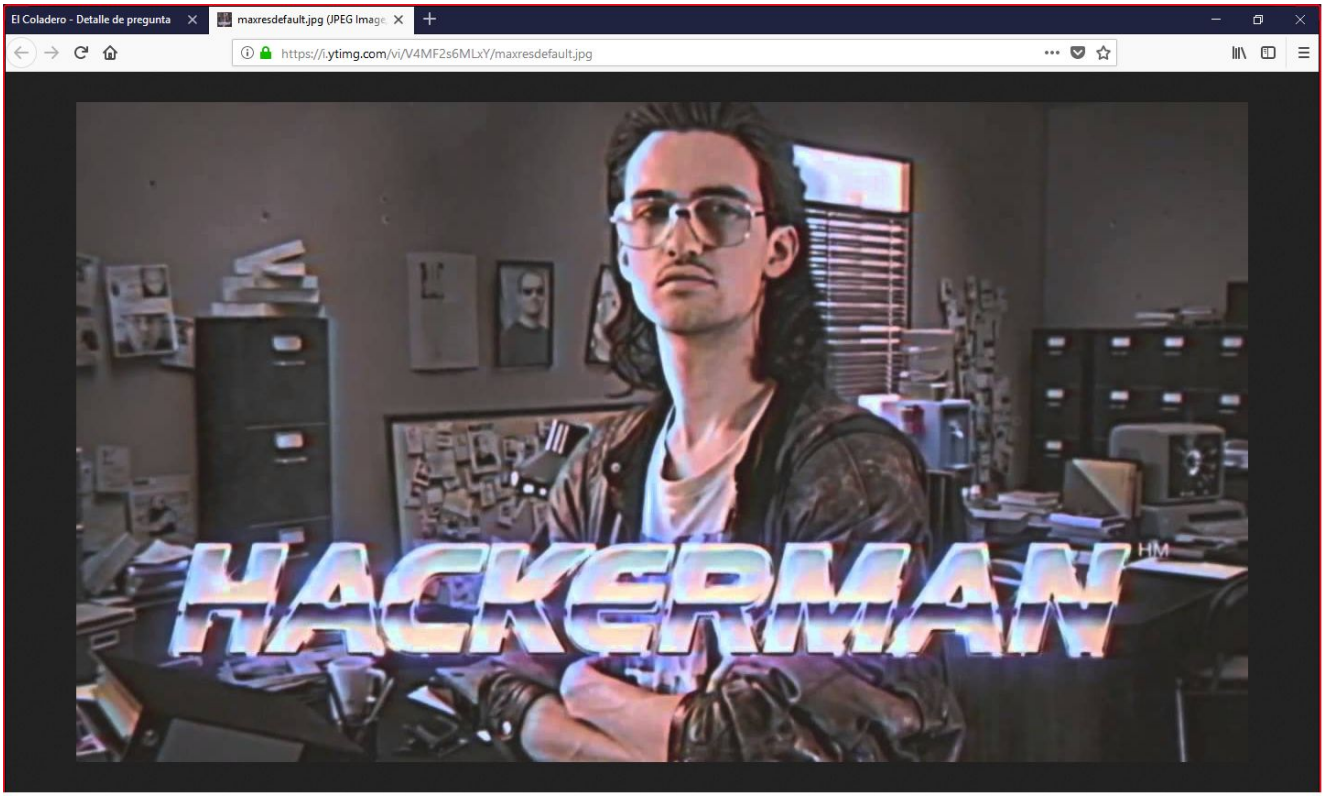
En este caso hemos usado el campo Cuerpo para introducir código javascript que al ser insertado en la respuesta se ejecutará al abrir esta página. En este caso hemos introducido un código que mostrará un mensaje y a continuación abrirá un enlace en una pestaña nueva.



Una vez insertada la respuesta, volvemos a la página inicial y hacemos clic en Ver de la pregunta anterior.



Observamos el mensaje mostrado.



Y al darle a OK se ejecuta lo siguiente, abrir este enlace en una pestaña nueva.

Medidas para mitigar la vulnerabilidad
Sanitizar la entrada de los campos del formulario de forma que escapen caracteres no deseados antes de pasarlo al template. Una opción puede ser usar la librería Bleach (pip install bleach) ya comentada en las medidas de mitigación del Reflected XSS y hacer “campo = bleach.clean(campo)” [Donde “campo” es el nombre de cada uno de los parámetros recibidos del formulario]. De esta forma al insertarlos en el html no se ejecutará el script malicioso ya que habremos escapado todos los caracteres problemáticos que pudiera haber.