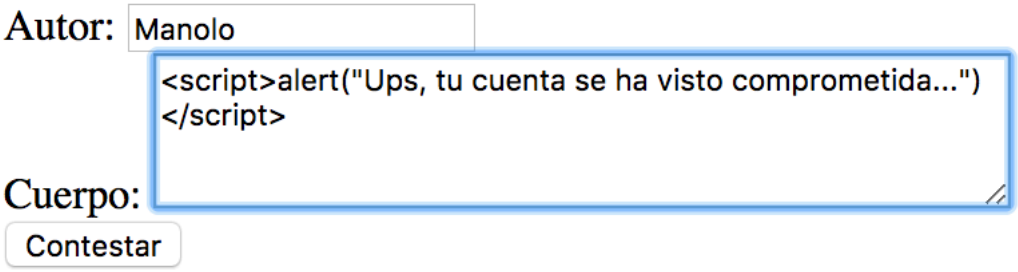
Víctor Fernández Rubio, David González Jiménez, Carlos Llames Arribas y Marta Pastor Puente declaramos que esta solución es fruto exclusivamente de nuestro trabajo personal. No hemos sido ayudados por ninguna otra persona ni hemos obtenido la solución de fuentes externas, y tampoco hemos compartido nuestra 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.

Gestión de la Información en la Web

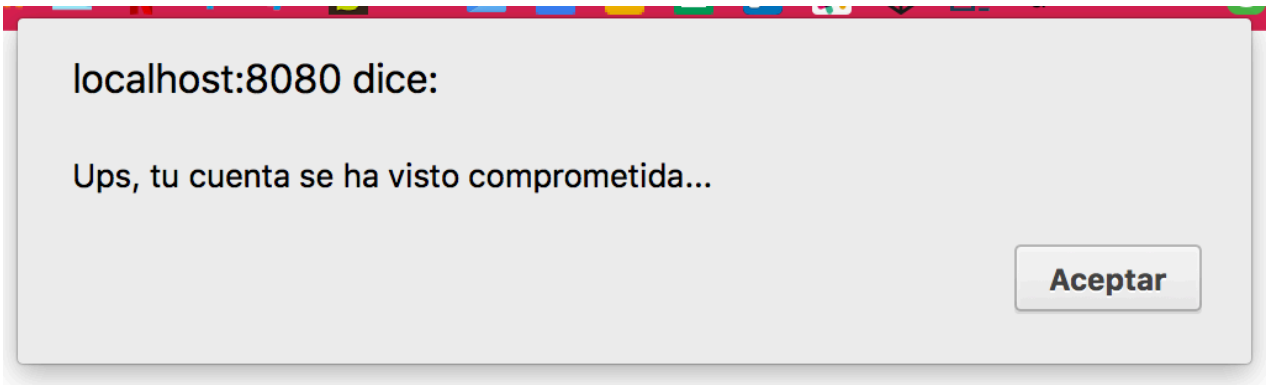
Práctica 12 - Auditoría de Seguridad

INFORME DE VULNERABILIDAD
Ruta(s) de la aplicación involucrada(s)
`@post('/insert_reply')`
Tipo de vulnerabilidad
XSS Persistente
Causante de la vulnerabilidad
Incorrecto tratamiento de los datos de entrada del formulario que se pasan como parámetros POST, permitiendo de esta forma insertar código malicioso de forma permanente en la base de datos al intentar una respuesta a una pregunta de un hilo, de forma que cada vez que entremos en dicha pregunta, se ejecute el código malicioso.
Situaciones peligrosas o no deseadas que puede provocar
Debido a que se puede modificar de manera permanente la base de datos o el código de la web que se carga desde el servidor, podrían incluirse etiquetas HTML como `img` o `iframe` con una URL maliciosa que redireccionara a otra web cada vez que se cargara la página o enviara todos los datos que el usuario introdujera en la misma para iniciar sesión a un máquina remota. Además, también puede darse el caso de ejecutar código Javascript malicioso que, al estar guardado en la base de datos como un campo de alguna de las preguntas, cada vez que quisiéramos mostrar dicha pregunta, el código ejecuta un SELECT sobre la tabla y devuelve la fila con el campo que contiene en código malicioso.
Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla)

Se ha detectado la vulnerabilidad a la hora de insertar una respuesta a un hilo ya abierto, lo que nos permite subir un código Javascript malicioso como, por ejemplo, el campo “Cuerpo” de la respuesta. Si por ejemplo para la pregunta “Listas en Python” subimos la siguiente respuesta al hilo del foro…



… cada vez que entremos desde la pantalla principal a la pregunta “Listas en Python”, nos aparece el siguiente mensaje emergente:



Este es un ejemplo muy básico hecho con un `alert()` que ejecutan los navegadores Chrome y Firefox. No obstante, si el foro contara con un mecanismo de inicio y autenticación de los usuarios, se podría explotar esta vulnerabilidad para que cada vez que se recargara la página, nos enviara las cookies y los datos de la sesión iniciada a nuestro servidor remoto.

Lo más curioso de esta vulnerabilidad, no obstante, es que si intentamos explotarla sin haber analizado previamente el código con el otro formulario que existe, el de crear una pregunta nueva para el foro, comprobamos que sin embargo la vulnerabilidad no tiene efecto ya que la forma de tratar los datos en Python es distinta, ya que /insert_question somete los datos de entrada del formulario a `format()` para impedir precisamente guardar código malicioso con etiquetas contenidas entre los símbolos < y >, y por ello la web acaba mostrando:

Título:	**Persepolis**
Autor:	Marjane Satrapi
Fecha:	2018-02-04 23:37:52
Etiquetas:	novela grafica, iran
Cuerpo:	<script>alert("Ups, tu cuenta se ha visto comprometida...")</script>

Medidas para mitigar la vulnerabilidad
Entre las medidas que se proponen, se encuentra limpiar siempre bien los datos de entrada de los formularios, de tal manera que evitemos la inserción de símbolos como < o >, convirtiendo estos en caracteres a la hora de guardarlos en la base de datos y volviendo a convertirlos a sus símbolos correspondientes únicamente en el momento de mostrar el texto en el navegador.

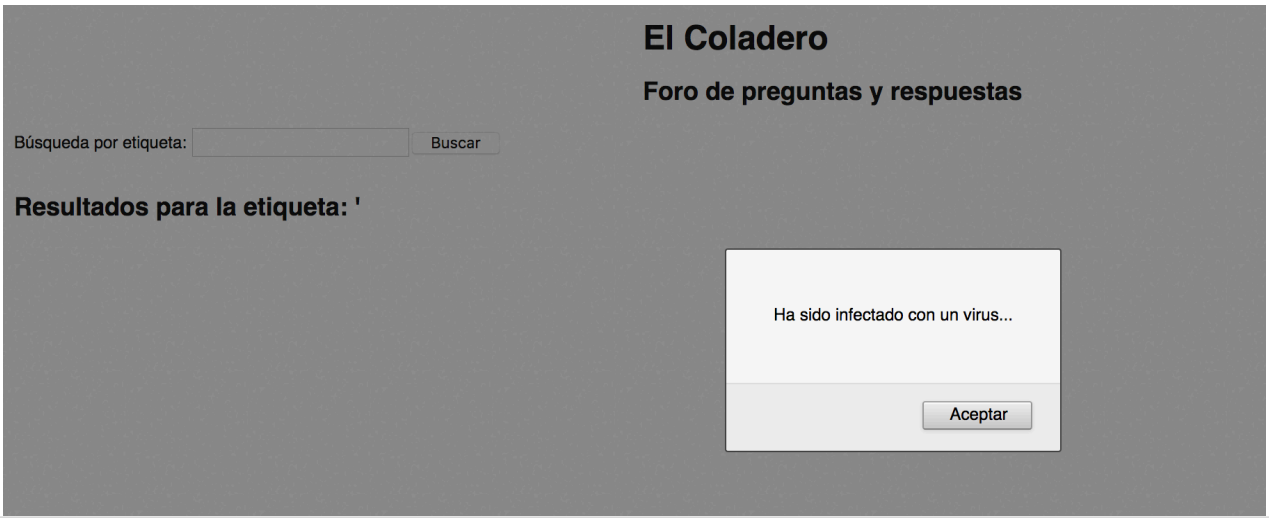
INFORME DE VULNERABILIDAD
Ruta(s) de la aplicación involucrada(s)
`@get('/search_question')`
Tipo de vulnerabilidad
XSS Reflejado
Causante de la vulnerabilidad
Incorrecto tratamiento de los datos pasados como parámetros de una función GET, de forma que al lanzar dicha función desde un enlace con el código malicioso pasado como argumento, dicho código se ejecuta en el ordenador de la víctima. En este tipo de vulnerabilidades, es necesarios que la víctima activa entre expresamente con el enlace infectado, ya que es una vulnerabilidad que afecta únicamente a la sesión y no a toda la aplicación.
Situaciones peligrosas o no deseadas que puede provocar
Al igual que ocurre con la vulnerabilidad anterior, esta permite ejecutar código malicioso que se pasa como parámetro a la llamada a la función /search_question, de tal forma que en vez de devolver los resultados de la consult, se ejecuta directamente el código malicioso. No obstante, a diferencia de lo que ocurría con la vulnerabilidad anterior, los cambios no son permanentes y sólo afectan a los usuarios que accedan a la URL que lleva incluido el código malicioso como parámetro de la llamada a la función de búsqueda, por lo que no se compromete la aplicación para todos sus usuarios sino únicamente para aquellos que accedan a través de la URL maliciosa.
Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla)

Lo primero que debemos hacer es escribir en el cuadro de búsqueda el código Javascript que queremos ejecutar directamente:

En este caso, queremos que se muestre una ventana emergente con el mensaje “Ha sido infectado con un virus…”. Cuando lo intentamos ejecutar la primera vez en Google Chorme, observamos que el navegador cuenta con medidas por defecto para detectar y evitar este tipo de llamadas GET maliciosas, porque al detectar la siguiente URL:

el navegador Chrome devuelve el siguiente mensaje de error, advirtiendo de que se ha detectado un intento de ataque XSS:

Sin embargo, si lo probamos con el navegador Firefox, comprobamos que efectivamente nuestro ataque surte efecto:

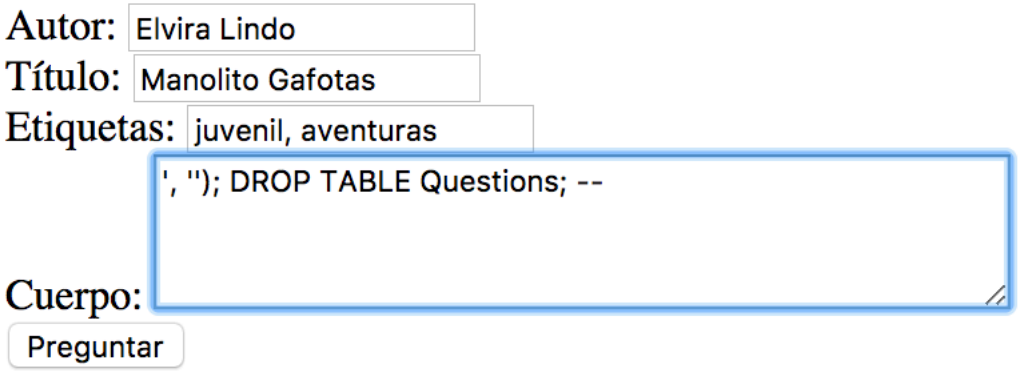


Medidas para mitigar la vulnerabilidad
Para evitar este tipo de ataques en navegadores que no tienen detección automática como ha sido el caso de Chrome, se recomienda hacer una limpieza previa antes de ejecutar la URL con los parámetros introducidos, comprobando primeramente que dichos parámetros sólo contienen los caracteres permitidos por nosotros (recomendablemente, letras mayúsculas, minúsculas y números del 1 al 9, por lo que no permitamos ningún símbolo) y tienen una longitud máxima de por ejemplo 20 caracteres.

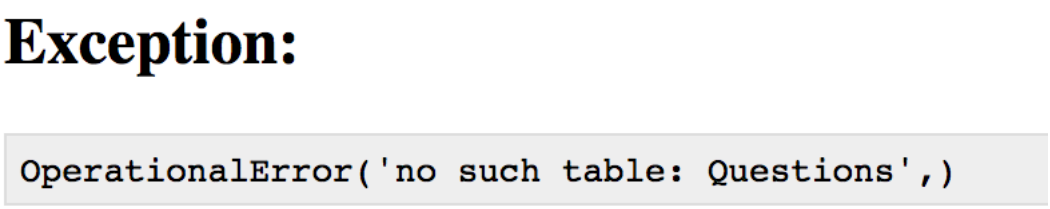
INFORME DE VULNERABILIDAD
Ruta(s) de la aplicación involucrada(s)
`@post('/insert_question')`
Tipo de vulnerabilidad
SQL Injection
Causante de la vulnerabilidad
Incorrecta programación de las funciones del servidor que permiten ejecución de más de una instrucción sobre la base de datos, pudiendo de esta forma modificar los datos de entrada de un formulario para así ejecutar código SQL sobre la base de datos sin autorización de la víctima.
Situaciones peligrosas o no deseadas que puede provocar
Permite al atacante modificar los datos que se envían como parámetros POST de la función de tal forma que finalicen la inserción de datos en la base de datos y ejecuten posteriormente otro tipo de consulta sobre dicha base que puede suponer el borrado o modificación de alguna o todas las tablas.
Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla)

Lo primero que deberíamos intentar es mostrar todas las tablas del engine (en nuestro caso es SQLite3, pero podría ser Oradle o cualquier otro) con una consulta del tipo `SELECT * FROM sqlite_master`. No obstante, como se trata de una inserción en la base de datos y ya hemos comprobado con anterioridad que no se pueden ejecutar código malicioso desde este formulario (ya que limpia correctamente los datos de entrada haciendo uso de la función `format()`) no podemos ejecutar dicha consulta.

Partimos de la base de que conocemos la existencia de la tabla Questions. En caso de no hacerlo, podríamos empezar a probar tablas que se nos ocurrieran que podrían existir por la estructura de la web (como podría ser Questions, Replies o en caso de contar con autenticación de usuarios, la tabla Users). Si ejecutamos el siguiente código en el formulario para enviar una nueva pregunta al foro…



… eliminamos directamente la tabla Questions por completo, de tal forma que al intentar acceder de nuevo a la ruta /show_all_questions, el navegador nos devuelve el siguiente mensaje de error:



Esta vulnerabilidad ocurre porque empleamos la función `executescript()` en el código Python de nuestro servidor en vez de utilizar sólo `execute()`, lo que permite ejecutar más de una consulta sobre la base de datos de una vez y por lo tanto, permite al atacante modificar filas o tablas sin nuestro consentimiento.

Medidas para mitigar la vulnerabilidad

Como se ha explicado ya, deberíamos emplear la función `execute()` para permitir así únicamente la ejecución de una instrucción SQL. Además, se deberían escapar los caracteres y símbolos especiales, sobre todos las comillas simples y los paréntesis, usando los propios argumentos de la función `execute()`.

También, en caso de que existiera un sistema de login, deberíamos diferenciar entre distintos tipos de usuarios, asignando distintos permisos de ejecución, lectura y modificación de las tablas de la base de datos en función de si son moderadores, administradores o usuarios normales, por ejemplo. De esta forma, nos aseguramos de que los usuarios normales sólo puede escribir en la tabla Questions pero no pueden realizar ninguna otra instrucción DDL como borrar una fila o la tabla por completo, acciones que deberían reservarse a los moderadores y al administrador respectivamente.

Por último, y este fallo es transversal a todas las rutas, deberíamos desactivar la opción de desarrollador en el código, ya que teniendo el parámetro `debug=True` lo que permitimos es que cualquier usuario pueda ver la traza del error lanzado por el servidor, dándose así más información de la necesaria. Este valor debería ser siempre False en los entornos de producción.