



## INFORME DE VULNERABILIDADES

### XSS - Stored

## 1.- INTRODUCCIÓN

El presente informe aborda las vulnerabilidades de XSS (Cros Site Scripting) en su tipo “Stored” en aplicaciones web, analizando dos ejemplos de código con niveles de seguridad bajo y medio donde se explica cómo se producen estas vulnerabilidades y se proporcionan recomendaciones para mitigarlas.

## 2. VULNERABILIDAD XSS – STORED (NIVEL BAJO)

### 2.1.- Descripción (XSS-Stored, persistente o tipo 2):

Es un tipo de ataque de inyección de código malicioso en sitios web que se almacena en el servidor y se ejecuta cada vez que un usuario visita la página infectada, sin necesidad que el atacante esté en conexión, por lo que puede afectar a un gran número de usuarios.

Un ataque XSS-Stored<sup>1</sup> funciona de la siguiente manera:

1. Un atacante inyecta un código malicioso en una aplicación web, normalmente a través de un formulario de entrada de usuario, como un comentario en un blog o un mensaje en un foro.
2. El código malicioso se almacena en la base de datos del sitio web.
3. Cuando un usuario visita la página infectada, el código malicioso se carga desde la base de datos y se ejecuta en el navegador del usuario.
4. El código malicioso puede acceder a la sesión del usuario, robar sus credenciales, redirección a otro sitio web o realizar otras acciones maliciosas.

---

<sup>1</sup>[https://owasp.org/www-project-web-security-testing-guide/v41/4-Web\\_Application\\_Security\\_Testing/07-Input\\_Validation\\_Testing/02-Testing\\_for\\_Stored\\_Cross\\_Site\\_Scripting.html](https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/07-Input_Validation_Testing/02-Testing_for_Stored_Cross_Site_Scripting.html)

## 2.2. Ejemplo de Código extraído de DVWA, de NIVEL BAJO<sup>2</sup>:

```
<?php
// verifica si se ha enviado el formulario a través del método POST antes de procesar los datos
if( isset( $_POST[ 'btnSign' ] ) ) {

    // Conexión a la base de datos y se selecciona la base de datos donde se almacenarán los mismos
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name     = trim( $_POST[ 'txtName' ] );

    // mediante la función stripslashes se eliminan las barras invertidas de un string en versiones antiguas de PHP
    // para evitar que se almacenen en la base de datos y se muestren en la página
    $message = stripslashes( $message );

    // uso de una condición ternaria que verifica si la variable es un objeto en la base de datos
    // si es así se escapa el string, y en caso contrario se muestra un mensaje de error
    $message = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"],
    $message ) : ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

    // Al igual que con la variable $message, se verifica si la variable $name es un objeto en la base de datos
    $name = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"],
    $name ) : ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

    // Se ejecuta la consulta SQL para insertar los datos en la base de datos, siendo escrita de manera manual.
    // Esto presenta un riesgo de inyección SQL, ya que no se están utilizando consultas preparadas
    // Finalmente se cierra la conexión a la base de datos, siendo almacenados los datos en la tabla guestbook
    $query = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '<pre>' . ((is_object($GLOBALS["__mysqli_ston"]))
    ? mysqli_error($GLOBALS["__mysqli_ston"]) : (($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );

}
?>
```

### 2.2.1.- Explicación:

Este código PHP está diseñado para procesar un formulario enviado, sanitizar los datos recibidos y luego insertarlos en una base de datos. A continuación, se analiza cada línea de código relevante:

1. **(php)** – inicio del código PHP.
2. **if( isset( \$\_POST[ 'btnSign' ] ) ) {** - Esta línea verifica si el botón del formulario con el nombre `btnSign` ha sido presionado y si es positivo, el código dentro del bloque `if` se ejecutará.

<sup>2</sup> Se ha mostrado de esta forma, ya que no cabía con una imagen de la fuente original

3. `$message = trim( $_POST[ 'mtxMessage' ] );` - Aquí se recupera el valor enviado desde el campo `mtxMessage` del formulario, se elimina cualquier espacio en blanco al principio y al final del texto y se almacena en la variable `\$message`.

3. `$name = trim( $_POST[ 'txtName' ] );` - Similar a la línea anterior, esta línea recupera y limpia el valor enviado desde el campo `txtName` del formulario, almacenándolo en la variable `\$name`.

4. `message = stripslashes( $message );` - Esta línea elimina las barras invertidas automáticamente de la variable `\$message`, que podrían haber sido añadidas para escapar caracteres especiales, lo cual es útil para restaurar el formato original de los datos que han sido alterados por el mecanismo de "magic quotes"<sup>3</sup> de PHP en versiones anteriores comillas.

5. `$name = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name) : ((trigger_error("[MySQLConverterToo] Fix the mysqli_real_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));` - Esta línea compleja realiza una operación condicional ternaria<sup>4</sup> para sanitizar la variable `\$message` usando `mysqli\_real\_escape\_string` para prevenir inyecciones SQL. Si la conexión global a la base de datos (`\$GLOBALS["\_\_mysqli\_ston"]`) está establecida y es un objeto de la misma, usará la función `mysqli_real_escape_string()` para escapar de la variable `\$name`. En caso contrario, la función `trigger_error` enviara un mensaje de error.

---

<sup>3</sup> función comúnmente utilizada para limpiar datos que han sido escapados automáticamente por PHP de versiones antiguas, para añadir automáticamente barras invertidas antes de caracteres que se consideran especiales en contextos de cadenas de texto, como las comillas simples ('), las comillas dobles ("), la barra invertida (\) y el carácter NULL encontrándose en las versiones nuevas en desuso.

<sup>4</sup> Es un operador condicional que se utiliza en programación para simplificar la estructura de un *condicional if-else* en una sola línea de código, siendo su estructura: [ condición ? valor si es verdadero : valor si es falso]. Es importante usarlo de manera entendible para el usuario.

6. `$name = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name) : ((trigger_error("[MySQLConverterToo] Fix the mysqli_real_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));` - La sanitización de la variable `$name` se realiza de manera similar

a ``$message``, utilizando ``mysqli_real_escape_string`` para escapar caracteres especiales y prevenir inyecciones SQL.

7. `query = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";` - Esta línea construye la consulta SQL para insertar los valores de ``$message`` y ``$name`` en la tabla ``guestbook``.

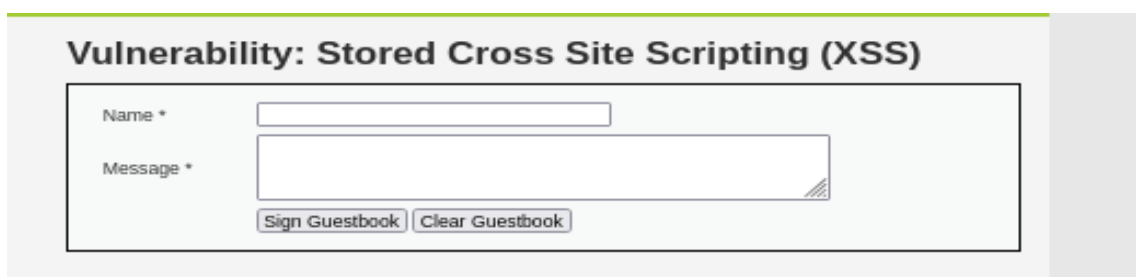
8. `$result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '' );` - Esta línea ejecuta la consulta SQL almacenada en ``$query`` usando la conexión global a la base de datos. Si la consulta falla, se detiene la ejecución del script y se muestra el error de MySQL.

9. `?>` - cierre del script PHP.

En resumen, este código recibe datos de un formulario, los sanitiza para prevenir inyecciones SQL, siendo esta su mayor vulnerabilidad, ya que, aunque sanitiza las entradas utilizando `"mysqli_real_escape_string"`, la forma en que se construye la consulta SQL (`$query`) con variables directamente incrustadas en la cadena, podría permitir a un atacante manipular `"mtxMessage"` o `"txtName"` para alterar la consulta SQL y ejecutar comandos maliciosos en la base de datos. Un uso más seguro y actualizado sería el uso de consultas preparadas para evitar inyecciones SQL, entre otras medidas que se analizarán a continuación.

### 2.2.2- Vulnerabilidad a XSS-Stored de la maquina DVWA

Como hemos explicado en el punto anterior, el código presenta deficiencias y sin protección ante ataques SQL mediante inserción de comandos, en caso de ataque XSS-persistente. Si analizamos la web DVWA, en el nivel bajo encontramos una caja de inputs donde se pueden introducir el nombre y el mensaje y dos botones, uno para enviar la consulta y otra para borrar todo lo escrito en el interior de la caja:



**Vulnerability: Stored Cross Site Scripting (XSS)**

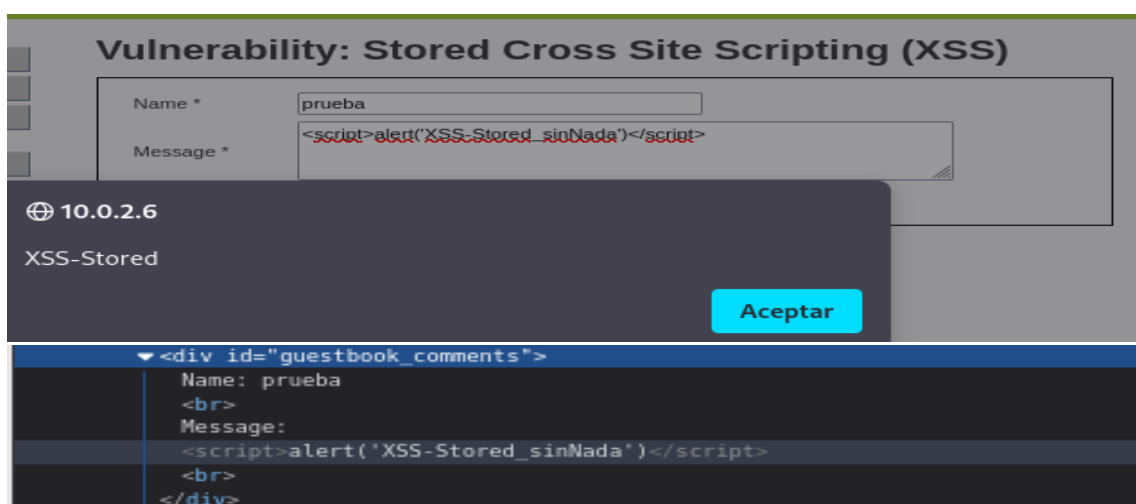
Name \*

Message \*

Se ejecuta código script directamente en el campo “message”:

```
<script>alert("XSS.Stored_sinNada")</script>
```

Obteniendo como resultado que es vulnerable a infección de código XSS y además de tipo persistente, ya que queda almacenado en el código, hasta que reseteo la maquina DVWA, como se puede observar:



**Vulnerability: Stored Cross Site Scripting (XSS)**

Name \*

Message \*

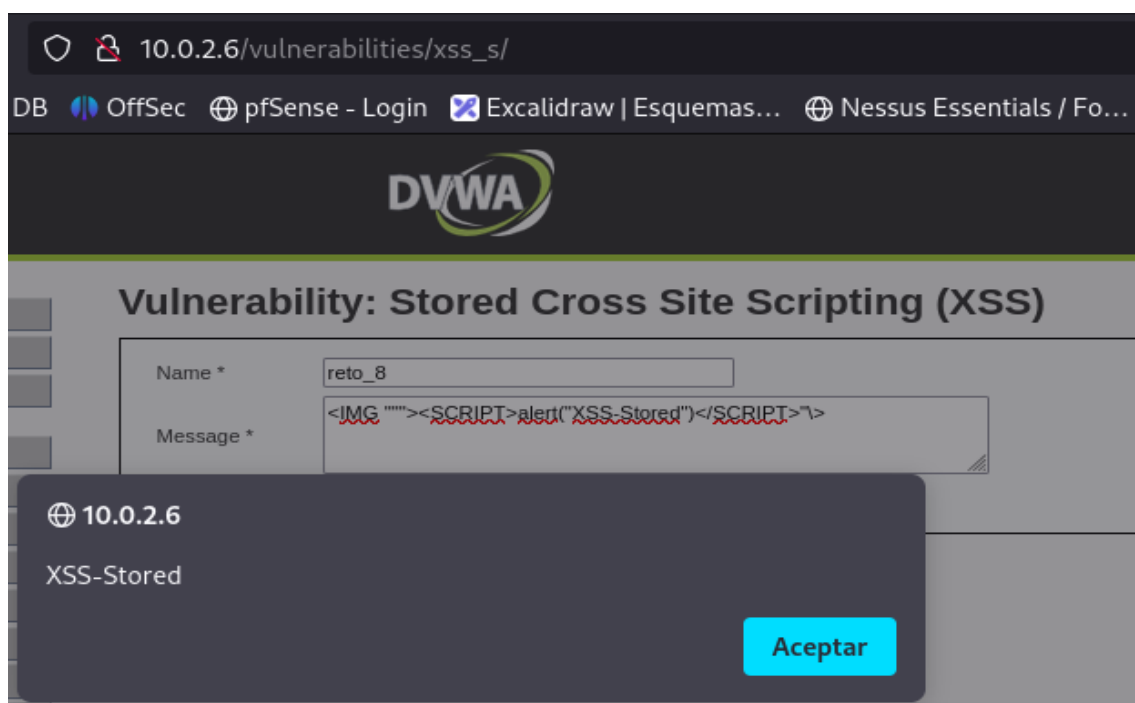
10.0.2.6  
XSS-Stored

```
<div id="guestbook_comments">  
  Name: prueba  
  <br>  
  Message:  
  <script>alert('XSS-Stored_sinNada')</script>  
  <br>  
</div>
```

Además, se ha probado ocultar el código script dentro de las etiquetas HTML de una imagen y ejecutarlo en el campo “message”:

```
<img ""=""><script>alert("XSS-Stored")</script>>
```

Obteniendo como resultado el mismo resultado que si se ejecuta código script directamente, es decir, al código le falta formas de escapar y sanear el código del usuario como ya se explico en el punto anterior:



```
<div id="guestbook_comments">
  Name: reto_8
  <br>
  Message:
  <img ""="">
  <script>alert("XSS-Stored")</script>
  >
  <br>
</div>
```



### 2.2.3. Ejemplo de Mejora de código para el nivel bajo

```
<?php

// Se verifica si se ha enviado el formulario a través del método POST antes de procesar los datos
if(isset($_POST['btnSign'])) {

    // Conexión a la base de datos y se selecciona la base de datos donde se almacenarán los mismos
    $mysqli = new mysqli("host", "usuario", "contraseña", "base_de_datos");

    // Se verifica si la conexión a la base de datos fue exitosa y en caso contrario se muestra un mensaje de error
    if($mysqli->connect_error) {
        die("Conexión fallida: " . $mysqli->connect_error);
    }

    // Se prepara la consulta SQL para insertar los datos en la base de datos, utilizando consultas preparadas
    $stmt = $mysqli->prepare("INSERT INTO guestbook (comment, name) VALUES (?, ?)");

    // Se vinculan los parámetros de la consulta preparada con las variables que contienen los datos a insertar
    $stmt->bind_param("ss", $message, $name);

    // Se obtienen los datos del formulario inicial y se eliminan los espacios en blanco al inicio y al final
    $message = trim($_POST['mtxMessage']);
    $name = trim($_POST['txtName']);

    // Se ejecuta la consulta preparada para insertar los datos en la base de datos y en caso de error se muestra un mensaje de error
    if(!$stmt->execute()) {
        // Manejar error de manera segura
        error_log("Error en la inserción: " . $stmt->error);
        echo "<pre>Error al procesar su solicitud. Por favor, intente de nuevo más tarde.</pre>";
    } else {
        echo "<pre>Mensaje guardado con éxito.</pre>";
    }

    // Se procede a cerrar la consulta preparada y la conexión a la base de datos
    $stmt->close();
    $mysqli->close();
}

?>
```

Este código mejorado utiliza consultas preparadas para evitar inyecciones SQL, mejora el manejo de errores sin exponer detalles sensibles al usuario y sigue las prácticas recomendadas para la conexión y ejecución de consultas en la base de datos.



## 2.3.- Ejemplo de Código extraído de DVWA, de **NIVEL MEDIO**

```
<?php

//Se verifica si se ha enviado el formulario a través del método POST antes de procesar los datos
if( isset( $_POST[ 'btnSign' ] ) ) {

    // Se selecciona la base de datos donde se almacenarán los datos y se eliminan los espacios en blanco al inicio y al final
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name     = trim( $_POST[ 'txtName' ] );

    // Se eliminan las barras invertidas de un string en versiones antiguas de PHP para evitar que se almacenen en la base de datos y se muestren en la página
    $message = strip_tags( addslashes( $message ) );

    // Se verifica si la variable $message es un objeto en la base de datos y mediante la función mysqli_real_escape_string se escapa el string,
    // y en caso contrario se muestra un mensaje de error
    $message = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message ) :
    ((trigger_error("MySQLConverterToo: Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

    // Se convierten los caracteres especiales en entidades HTML de texto plano para evitar ataques XSS
    $message = htmlspecialchars( $message );

    // Se sanitiza el nombre para evitar ataques XSS y se eliminan las barras invertidas de un string en versiones antiguas de PHP
    $name = str_replace( '<script>', '', $name );

    // Se verifica si la variable $name es un objeto en la base de datos y mediante la función mysqli_real_escape_string se escapa el string,
    // y en caso contrario se muestra un mensaje de error
    $name = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name ) :
    ((trigger_error("MySQLConverterToo: Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

    // Se crea la consulta SQL para insertar los datos en la base de datos, siendo escrita de manera manual
    // Esto presenta un riesgo de inyección SQL, ya que no se están utilizando consultas preparadas
    $query = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";

    // Se ejecuta la consulta SQL realizada anteriormente y en caso de error se muestra un mensaje de error, cerrando la conexión a la base de datos
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '<pre>' . ((is_object($GLOBALS["__mysqli_ston"])) ? mysqli_error($GLOBALS["__mysqli_ston"]) :
    (($__mysqli_res = mysqli_connect_error()) ? $__mysqli_res : false)) . '</pre>' );
}

}>
```

### 2.3.1.- Explicación:

1. Inicio y cierre del documento PHP.
2. Los primeros pasos son similares al código de nivel bajo: primero se verifica el envío del formulario cumplimentado, comprobando si el botón “btnSign” están incluido en el mismo, después se obtiene los valores de las variables \$message y \$name, eliminándose los espacios en blanco al inicio y al final de cada valor.

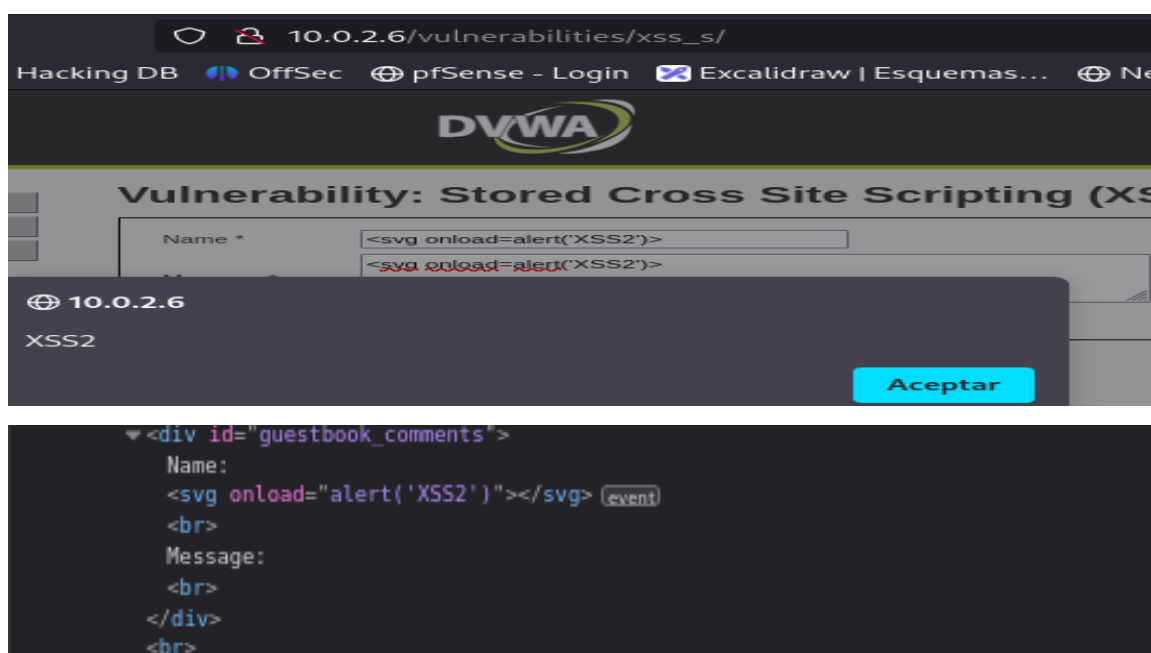
3. `$message = strip_tags( addslashes( $message ) );` En esta parte si hay cambios respecto al nivel bajo, aquí se eliminan las etiquetas HTML de la variable `$message` y se añaden barras invertidas a las comillas del mensaje.
4. `$message = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_sto"]))) --` Se verifica en una condición, si la variable `GLOBALS` está definida (*isset*) y si la variable `GLOBALS` es un objeto, devolviendo en caso afirmativo `True` y en el caso contrario `"null"`, es decir, devolverá `True` siempre y cuando la variable `"GLOBALS__mysqli_ston"` este definida y sea un objeto de la BBDD MySQL.
5. `? mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message ) : --` Con el operador ternario (?) comienza una línea de comprensión, donde se establece como condición el cumplimiento de lo establecido en el punto 4, si ambos son `True` se ejecutara el código , siendo sanitizado por la función `"mysqli_real_escape_string"`, la cual escapará todos los caracteres especiales para uso de una consulta SQL, usando la conexión a la BBDD `"GLOBALS"` para ello, ayudando la prevención de inyecciones SQL, finalizando con la otra parte del operador ternario (:) , la cual se ejecutará a partir de aquí, si la condición es `"False o Null"`.
6. `((trigger_error("[MySQLConverterToo] Fix the mysqli_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : "");` -- Se genera un error por la herramienta `"MySQLConverterToo"`, que se usa para la conversión de las funciones del código antiguo MySQL a la nuevas funciones de MySQLi, señalando que, en caso que se den estas funciones obsoletas, sen reemplazadas por la alternativa moderna o por sentencias preparadas, siendo todas estas mucho mas seguras.
7. `$message = htmlspecialchars( $message );` -- Dentro del valor de la variable `$message` , se convierte los caracteres especiales a entidades HTML en texto plano, evitando el peligro de ejecución de scripts maliciosos.
8. `$name = str_replace( '<script>', '', $name );` -- Se elimina cualquier mención u aparición de la etiqueta `"<script>"` en la variable `$name`.

9. Las tres líneas de código ultimas, son idénticas a las explicadas en el código de nivel bajo, es decir, primero se ejecuta un saneamiento en una condición ternaria de la variable `$name`, usando la función `'mysqli_real_escape_string'` para escapar caracteres especiales y prevenir inyecciones SQL, para seguidamente crear la consulta SQL (*query*) insertando los valores de las variables en la tabla *"guestbook"* y finalmente se ejecuta la consulta almacenada en la `$query` usando la conexión *GLOBALS* a las BBDD, devolviendo si es correcta, y si es errónea , detendrá la ejecución del script.

### 2.3.2- Vulnerabilidad a XSS-Stored de la maquina DVWA

Si analizamos la web DVWA, al igual que en el nivel bajo al introducir el nombre este y mensaje se almacena en el navegador de la web. El código ahora tiene más protecciones:

- En la variable `$message` se eliminan todas las etiquetas HTML y PHP, se añaden barras invertidas a las comillas simples y dobles y a los caracteres NULL, sanea los caracteres espaciales de una cadena de uso SQL y convierte estos en entidades HTML de texto plano.
- En la variable `$name`, elimina cualquier aparición de la etiqueta `<script>` y sanea los caracteres especiales para uso en una consulta SQL.



Teniendo en cuenta la información anterior, es más fácil atacar la variables name, al tener menos medidas de protección contra XSS, por lo que se introduce en el campo name y message: “<svg onload=alert('XSS')>”, donde usamos una etiqueta HTML acompañado de un atributo que carga la alerta XSS, esto hará que se almacene en la BBDD, y cuando se muestre el contenido HTML en la web se cargara la alerta XSS, sin embargo, en el campo message, queda atrapado dicho código por el resto de medidas de protección ya comentadas.

### 2.3.3.- Ejemplo de Mejora de código para el nivel medio

```
<?php

// Implementación de CSP para mejorar la seguridad contra ataques XSS
header("Content-Security-Policy: default-src 'self'; script-src 'self' https://trustedscripts.example.com; object-src 'none';
style-src 'self' https://trustedscripts.example.com; img-src 'self'; media-src 'none'; frame-src 'none'; font-src 'self';
connect-src 'self'; plugin-types 'none'; base-uri 'self';");

// Se verifica si se ha enviado el formulario a través del método POST antes de procesar los datos
if( isset( $_POST[ 'btnSign' ] ) ) {

    // Se selecciona la base de datos donde se almacenarán los datos y se eliminan los espacios en blanco al inicio y al final
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name = trim( $_POST[ 'txtName' ] );

    // Se eliminan las etiquetas HTML y las barras invertidas de un string en versiones antiguas de PHP para evitar que se almacenen en la base de datos
    $message = strip_tags( addslashes( $message ) );

    // Se verifica si la variable $message es un objeto en la base de datos y mediante la función mysqli_real_escape_string se escapa el string,
    // y en caso contrario se muestra un mensaje de error
    $message = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message ) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

    // Se convierten los caracteres especiales en entidades HTML de texto plano para evitar ataques XSS
    $message = htmlspecialchars( $message, ENT_QUOTES, 'UTF-8' );

    // Se sanitiza el nombre para evitar ataques XSS y se eliminan las barras invertidas de un string en versiones antiguas de PHP
    $name = str_replace( '<script>', '', $name );

    // Se verifica si la variable $name es un objeto en la base de datos y mediante la función mysqli_real_escape_string se escapa el string,
    // y en caso contrario se muestra un mensaje de error
    $name = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name ) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

    // Se crea la consulta SQL para insertar los datos en la base de datos, utilizando consultas preparadas para evitar inyecciones SQL
    $stmt = $GLOBALS["__mysqli_ston"]->prepare("INSERT INTO guestbook ( comment, name ) VALUES (?, ?)");
    if ($stmt) {
        $stmt->bind_param("ss", $message, $name);
        $stmt->execute();
        $stmt->close();
    } else {
        // Manejo de errores seguro
        error_log("Error en la preparación de la consulta: " . $GLOBALS["__mysqli_ston"]->error);
        echo "Ha ocurrido un error. Por favor, inténtelo de nuevo más tarde.";
    }
}

// Se cierra la conexión a la base de datos
mysqli_close($GLOBALS["__mysqli_ston"]);

?>
```

Este código mejorado, implementa una política de seguridad de contenido restrictiva ante numerosas situaciones en las que únicamente tendrá permiso el servidor origen y bloqueará todos los demás, lo que reduce significativamente el riesgo de ataques XSS. Además, utiliza la función `htmlspecialchars` para escapar correctamente la entrada del usuario, junto con `ENT_QUOTES` que los convierte en entidades HTML de texto plano, la función `strip_tags` junto a `addslashes` que elimina las etiquetas HTML y todos los espacios al inicio y final del mensaje, el uso de consultas preparadas y de sentencias ternarias, aumentando las medidas de saneamiento y escape para evitar ataques XSS-Stored.

### 3.- MITIGACIÓN DE RIESGOS FRENTE ATAQUES XSS-STORED

Para mitigar los efectos de esta vulnerabilidad, es importante validar y sanitizar todas las entradas del usuario, con alguna/s de las siguientes recomendaciones:

#### 3.1.- Host Side:

- Validación de entradas, asegurándose que todas las entradas de usuario sean válidas y esperadas, siendo recomendable el uso de listas blancas para permitir solo ciertos tipos de datos.
- Saneamiento de entradas, escapando o eliminando caracteres especiales que puedan ser utilizados para inyectar scripts usando funciones de saneamiento específicas para el contexto en el que se utilizarán los datos (HTML, JavaScript, URL, etc.).

```
<?php
// Se verifica si el email es valido, retornando true si lo es y false en caso contrario
function validateEmail($email) {
    return filter_var($email, FILTER_VALIDATE_EMAIL);
}
// se verifica si el número es un entero, retornando true si lo es y false en caso contrario
function validateInteger($number) {
    return filter_var($number, FILTER_VALIDATE_INT);
}
// se sanitiza la entrada de datos, usando la funcion htmlspecialchars para escapar caracteres especiales
// y ENT_QUOTES para convertir comillas dobles y simples en entidades HTML de texto plano
function sanitizeInput($input) {
    return htmlspecialchars($input, ENT_QUOTES, 'UTF-8');
}
// Se sanitiza las entradas $conn y $input con la funcion mysqli_real_escape_string
//para evitar ataques XSS e inyección SQL
function sanitizeForSQL($conn, $input) {
    return mysqli_real_escape_string($conn, $input);
}
```

ejemplo completo validación y saneamiento PHP (1/2)

```

}
// Se verifica si se ha enviado el formulario a través del método POST antes de procesar los datos
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Se establece la conexión a la base de datos con el servidor, nombre de usuario, contraseña y database
    $conn = new mysqli("localhost", "username", "password", "database");
    // Se comprueba que las variables del formulario existan y han sido enviadas a través del método POST
    $email = $_POST['email'];
    $age = $_POST['age'];
    $comment = $_POST['comment'];
    // se validan finalmente los datos, usando la función de sanitizeForSQL para evitar inyección SQL
    if (validateEmail($email) && validateInteger($age)) {
        $email = sanitizeForSQL($conn, $email);
        $age = sanitizeForSQL($conn, $age);
        $comment = sanitizeInput($comment);
    }
    // Se ejecuta la consulta SQL para insertar los datos en la base de datos, utilizando consultas preparadas
    $query = "INSERT INTO users (email, age, comment) VALUES ('$email', '$age', '$comment')";
    if ($conn->query($query) === TRUE) {
        echo "Nuevo registro creado con éxito";
    } else {
        echo "Error: " . $query . "<br>" . $conn->error;
    }
} else {
    echo "Datos inválidos";
}
// Se cierra la conexión a la base de datos
$conn->close();
}
?>

```

ejemplo completo validación y saneamiento PHP (2/2)

- Escapar de todas las salidas de datos que se muestran en la página web, usando funciones de escape adecuadas para cada contexto (HTML, JavaScript, CSS, PHP, etc.).

```

function escape_output($data) {
    return htmlspecialchars($data, ENT_QUOTES, 'UTF-8');
}

echo escape_output($user_input);

```

Ejemplo PHP

```

function escapeHTML(str) {
    var div = document.createElement('div');
    div.appendChild(document.createTextNode(str));
    return div.innerHTML;
}

document.getElementById('output').innerHTML = escapeHTML(userInput);

```

Ejemplo JavaScript

- Implementación de una política de seguridad de contenido (CSP) para restringir las fuentes de scripts y otros recursos, ayudando a mitigar el impacto de un ataque XSS al limitar la capacidad del atacante para ejecutar scripts maliciosos:

```

<?php
http-equiv="Content-Security-Policy"
content=" default-src 'self'; script-src 'self' https://trustedscripts.example.com; object-src 'none';
style-src 'self' https://trustedstyles.example.com; base-uri 'self'; form-action 'self';" />

```

Ejemplo código completo

- Configurar las cookies con las banderas (flags), HttpOnly y Secure, para evitar que sean accesibles desde JavaScript y asegurar que solo se envíen a través de conexiones HTTPS.
- Almacena los datos de usuarios de forma segura y sanitiza cualquier dato antes de almacenarlo en la base de datos.
- Uso de frameworks y bibliotecas que proporcionen mecanismos integrados para prevenir XSS, como plantillas de escape automático. Algunos ejemplos pueden ser Angular y React, entre otros.

### 3.2.- Client Side:

- Asegurar que cualquier dato que se inserte en el DOM (Document Object Model) a través de JavaScript es correctamente escapado.
- Uso de “*InnerText o textContent*” para insertar texto en el DOM, ya que estos no interpretan HTML, sino texto plano, evitando la ejecución de código malicioso, que, si utiliza la función *InnerHTML*, no lo evita, pudiendo, en este último caso, ejecutarse código XSS-Stored
- Aunque es una medida del lado del servidor, las políticas de seguridad de contenido (CSP) también protegen el lado del cliente al restringir la ejecución de scripts no autorizados.
- Mantén el navegador y todas las bibliotecas de JavaScript actualizadas para protegerte contra vulnerabilidades conocidas.
- Uso de frameworks o librerías de frontend que proporcionen mecanismos de escape y protección contra XSS, como Angular o React

Características	React	Angular
Naturaleza	Biblioteca	Framework completo
Lenguaje	JavaScript (JSX)	TypeScript
Enfoque de Datos	Unidireccional	Bidireccional
Filosofía	Declarativo	Declarativo e imperativo
Ecosistema	Flexible, requiere decisiones	Opinión fuerte, herramientas integradas
Gestión del Estado	Necesita bibliotecas adicionales	Incluye herramientas integradas
Rendimiento	Virtual DOM	Change Detection
Comunidad y Soporte	Amplia comunidad, muchas herramientas	Amplia comunidad, herramientas integradas
CLI	No oficial, pero hay opciones	Angular CLI



## 4.- RECOMENDACIONES MÁS IMPORTANTES

- Validar y sanear todas las entradas del usuario, asegurando que solo se puedan incluir archivos permitidos.
- Usar las cabeceras seguras de CSP, pudiendo incluir que solo se permita código JavaScript desde la fuente, rechazando cualquier otro que provenga no provenga de este.
- El uso de la función “*die ()*” para manejar errores de MySQL puede acarrear problemas de seguridad, ya que éste, muestra el mensaje de error directamente al usuario final, que puede contener información potencialmente sensible al usuario final y valiosa para un atacante como:
  - Detalles de la consulta (cual fue el fallo, como es la estructura de la BBDD, nombres de tablas y campos, etc)
  - Información del servidor, como versión de MySQL y el sistema operativo.
- En lugar de mostrar errores de MySQL directamente al usuario, registrar estos errores de manera segura en el servidor y que se muestren el usuario mensajes de error genéricos. Una mejora
- Usar antivirus o similar y mantenerlos actualizados.
- Tener todas las aplicaciones actualizadas, especialmente los navegadores web.
- Usar el “Output Encoding”, proceso que convierte caracteres especiales en una forma segura antes de su representación en la web, usando en los datos de salida del usuario métodos como el HTML Encoding, previniendo que el navegador interprete el contenido del usuario como código de la web.
- Uso de herramientas que ayuden a la detección y bloqueo de esto ataques (WAF, Anti-XSS, etc)
- Habilitar el “HttpOnly” en las cookies que contienen información importante o sensible, impidiendo que el su contenido sea accesible por JavaScript, reduciendo el riesgo de XSS.
- Configurar el servidor para restringir el acceso a archivos sensibles, asegurando que los archivos críticos no sean accesibles desde la web.

- Uso del HTTPs como mecanismo de comunicación entre navegador y servidor. ayudando a prevenir que los ataques XSS intercepten y modifiquen las comunicaciones.
- Uso de funciones como “Htmlspecialchars” o “ENT\_QUOTES” para mantener la entrada controlada, no dejando escapar ningún carácter que pueda ser usado para un ataque de inyección de código.

## 5.- CONCLUSIONES

Después de analizar los ataques XSS-Stored y las medidas de seguridad para prevenirlos, se puede concluir que la protección contra estos ataques requiere una combinación de prácticas efectivas de los riesgos y concienciación en todos los eslabones de la cadena, desde desarrolladoras, data Scientist, analistas de ciberseguridad, etc a la hora de creación de código y estar en una continua actualización de las vulnerabilidades emergentes. A continuación, se presentan los puntos más importantes para recordar:

- ❖ La validación y sanitización de entradas son fundamentales para su prevención, debiendo asegurarse que las entradas del usuario se ajusten a las reglas y patrones esperados y así, eliminen cualquier carácter o secuencia de caracteres que no sean necesarios o que puedan ser utilizados para inyectar código JavaScript.
- ❖ La política de seguridad de contenido (CSP) es un mecanismo de seguridad que permite a los desarrolladores definir qué fuentes de contenido web son seguras para una aplicación, ayudando a prevenir que los navegadores carguen contenido malicioso.
- ❖ Manejo adecuado de los errores, almacenándolos en un solo lugar del servidor y el uso de mensajes genéricos destinados al usuario.
- ❖ Asegurarse que las cookies que contienen información sensible (inicios de sesión, etc) estén marcadas con el atributo HttpOnly, impidiendo que el contenido de la cookie sea accedido por JavaScript, reduciendo el riesgo de estos ataques.

- ❖ Implementar medidas de educación y concienciación entre los desarrolladores y otros miembros del equipo, para que comprendan los riesgos y cómo prevenirlos.
- ❖ Realizar revisiones periódicas del código fuente, para identificar cualquier vulnerabilidad XSS-Stored, usando herramientas de análisis de código estático y dinámico para ayudar en este proceso.

En resumen, la seguridad total no existe, y concretamente en ciberseguridad, está en constante cambio, lo que obliga a sus profesionales a estar actualizados diariamente frente a las amenazas emergentes y estar siempre en alerta debido a la gran cantidad de ataques que se dan, de los cuales muchos son falsos positivos provocados por la activación de una hiperseguridad necesaria en los medios físicos y lógicos instalados por las corporaciones para su protección cibernética. En este punto, La IA puede ser una herramienta valiosa en la lucha contra los ciberataques, en general, y contra los XSS-Stored en particular, ya que, pueden ayudar en la automatización de herramientas que permitan la identificación, detección y análisis de patrones maliciosos, ayudando a los profesionales de la ciberseguridad contra los falsos positivos, así como en la realización de simulacros de ataques, que podría ejecutar la IA de manera automática mostrando finalmente su resultado.