

**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 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.

2.2. Ejemplo de Código extraído de DVWA, de **NIVEL BAJO**:



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**.**

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*"[[1]](#footnote-1) de PHP en versiones anteriores comillas.

5.  $name = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"],  $name ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : "")); - Esta línea compleja realiza una operación condicional ternaria[[2]](#footnote-2) para sanitizar la variable *$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.

6.$name = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"],  $name ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : "")); - La sanitización de la variable *$name* se realiza de manera similar a `$message`, utilizando `mysqli\_real\_escape\_string` para escapar caracteres especiales y prevenir inyecciones SQL.

1. 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"],  $query ) or die( '<pre>' . ((is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_error($GLOBALS["\_\_\_mysqli\_ston"]) : (($\_\_\_mysqli\_res = mysqli\_connect\_error()) ? $\_\_\_mysqli\_res : false)) . '</pre>' ); - Esta línea ejecuta la consulta SQL almacenada en `$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 y luego intenta insertar esos datos en una base de datos, con el uso de variables globales y la construcción manual de la consulta SQL pueden presentar riesgos de seguridad. Un uso más seguro y actualizado sería el uso de consultas preparadas para evitar inyecciones SQL.

Este código PHP está diseñado para recibir datos de un formulario a través del método POST, procesar esos datos y luego insertarlos en una base de datos. A continuación, se analiza el código en términos de seguridad y funcionalidad, identificando vulnerabilidades y ofreciendo recomendaciones para mejorar la seguridad.

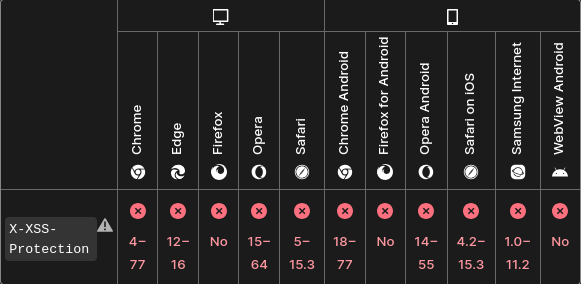
### Análisis de Seguridad y Vulnerabilidades

1. **Inyección SQL:** La mayor vulnerabilidad en este código es la inyección SQL. Aunque intenta sanitizar las entradas utilizando mysqli\_real\_escape\_string, la forma en que se construye la consulta SQL ($query) con variables directamente incrustadas en la cadena hace que el código sea susceptible a inyecciones SQL. Un atacante podría manipular mtxMessage o txtName para alterar la consulta SQL y ejecutar comandos maliciosos en la base de datos.
2. **Escapado de Entrada Ineficaz:** El uso de stripslashes y mysqli\_real\_escape\_string no es suficiente para garantizar la seguridad de las entradas. stripslashes se utiliza para eliminar las barras invertidas, lo cual puede ser parte de un esfuerzo para deshacerse de los efectos de addslashes o similar, pero no es una medida de seguridad efectiva por sí sola.
3. **Errores de MySQL Expuestos al Usuario:** El uso de die() para manejar errores de MySQL expone información potencialmente sensible al usuario final. Esto podría ser utilizado por un atacante para obtener información sobre la estructura de la base de datos o el entorno del servidor.

### Recomendaciones para Mejorar la Seguridad

1. **Preparar Consultas SQL:** Utilizar consultas preparadas con mysqli para evitar inyecciones SQL. Las consultas preparadas aseguran que los valores de entrada se manejen adecuadamente antes de ejecutar la consulta.
2. **Validación y Sanitización de Entradas:** Además de escapar las entradas, es importante validarlas contra un conjunto de reglas esperadas (por ejemplo, asegurarse de que un nombre solo contenga caracteres alfabéticos).
3. **Manejo de Errores Seguro:** En lugar de mostrar errores de MySQL directamente al usuario, registre estos errores de manera segura en el servidor y muestre mensajes de error genéricos al usuario.





**2.2.2- Vulnerabilidad a XSS-Reflected de la maquina DVWA**

Este código tiene una vulnerabilidad de seguridad grave a ataques XSS, debido a la forma en que se maneja el parámetro *“name*”, al concatenar la entrada del usuario en la salida del código, sin ningún tipo de validación o escape, permitiendo que un atacante pueda proporcionar un valor de *“name”* que incluya código HTML o JavaScript malicioso, lo que podría ser ejecutado en el navegador del usuario y permitir el robo de credenciales, inyección de malware u otros tipos de ataques.

**2.2.4. Ejemplo de Mejora de código para el nivel bajo**

****

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**

****

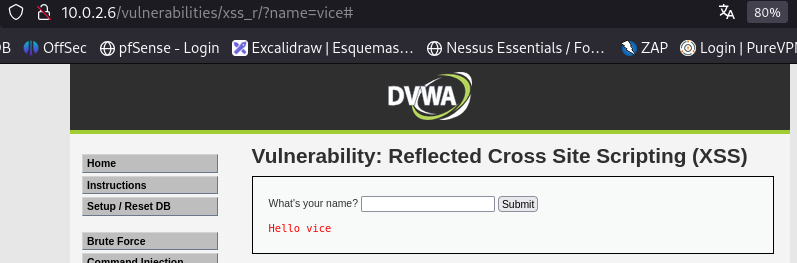
**2.3.1.- Explicación:**

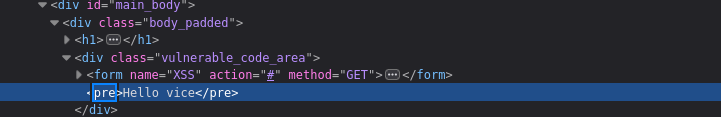
1. Inicio y cierre del documento PHP
2. Al igual que en el nivel bajo se inicia con la cabecera header ("X-XSS-Protection: 0"), lo que establece que el filtro de protección contra ataques XSS esta desactivado, pero como ya se ha comentado anteriormente, este método de seguridad ha quedado obsoleto, aplicándose en la actualidad formas más sofisticados y que aportan mayor seguridad, tanto a nivel de navegadores web, como en la implementación de las políticas de seguridad de contenidos (CSP).
3. $name = str\_replace( '<script>', '', $\_GET[ 'name' ] ); El código intenta prevenir ataques XSS reemplazando la cadena *`<script>`* por nada *,”,* en la entrada del usuario, siendo método extremadamente ineficaz y fácil de evadir, ya que un atacante podría utilizar muchas otras técnicas para inyectar código malicioso, como variaciones en el etiquetado *(<scripT> o <ScRiPt>*), eventos JavaScript que no contienen esa etiqueta *(<img src="x" onerror="alert('XSS')"*>), uso de entidades de HTML que represente a caracteres especiales*(<script> 🡪 &lt;script&gt;),* o uso de JavaScript en atributos HTML *(<a href="javascript:alert('XSS')">Click me</a>)*
4. echo "<pre>Hello ${name}</pre>“;} , como el código de nivel bajo, imprime directamente la entrada del usuario en la página, lo que hace que el código sea vulnerable a inyecciones HTML/JavaScript.

En resumen, este código PHP intenta manejar una entrada de usuario a través del parámetro “*name*” en la URL, aplicando una forma muy básica de saneamiento y validaciones de entradas, para prevenir ataques de Cross-Site Scripting (XSS); presentando varias deficiencias en términos de seguridad y prácticas de codificación.

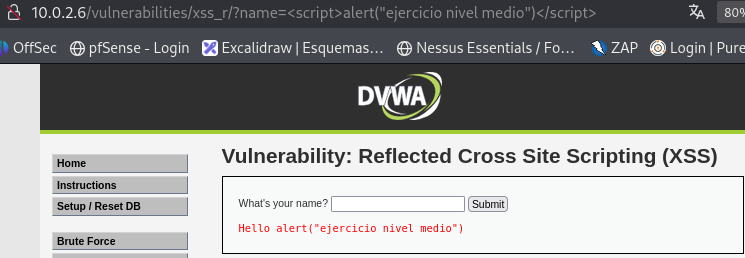
**2.3.2- Vulnerabilidad a XSS-Reflected de la maquina DVWA**

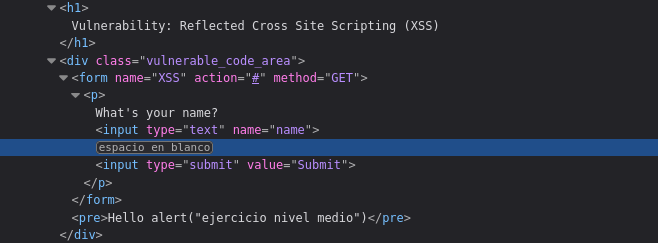
Como hemos explicando en el punto anterior, el código presenta deficiencias y una capa de seguridad muy básica para la protección de ataques XSS. Si analizamos la web DVWA, al igual que en el nivel bajo al introducir el nombre este aporta el saludo inicial incorporándose al código de la web.



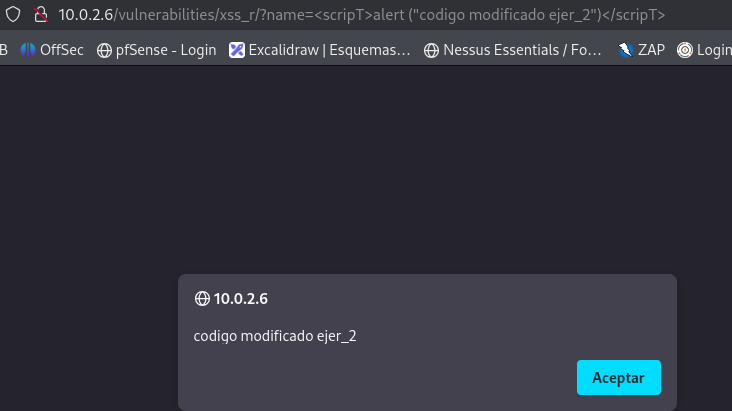


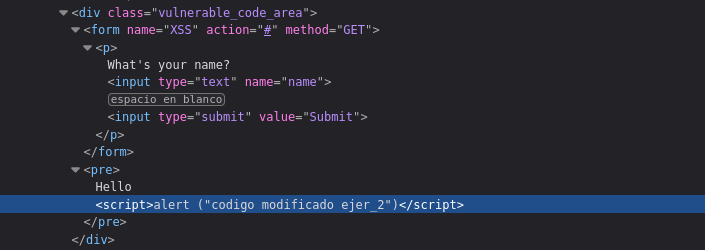
Pero, en caso de introducir código JavaScript directamente en la web como se puede ver en la imagen, además de no ejecutarse la ventana de alerta, tampoco aparece en el código de la pagina web, ya que tiene implementado un filtro con “*replace”* para evitar estas etiquetas.





Ahora, si ejecutamos el código java script usando variaciones en el etiquetado, observamos, que la medida de protección es salvada, ejecutándose el código malicioso e incorporándose al código de la aplicación web.



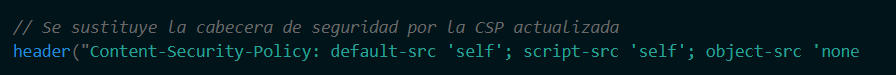


En definitiva, se comprueba que este entorno de prueba, es vulnerable a ataques XSS-Reflected, siendo necesario resaltar que en un entorno de producción debería haber una fase anterior, donde se usan técnicas de ingeniería social.

**2.3.3.- Mitigación del riesgo**

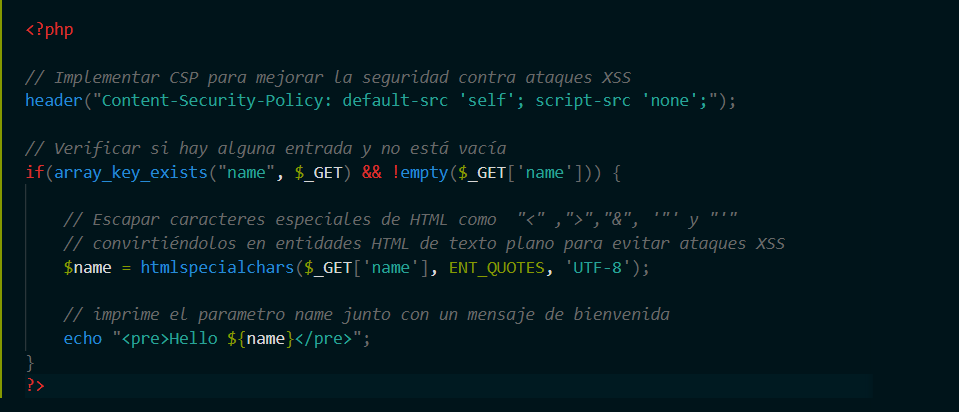
Para mitigar los efectos de esta vulnerabilidad, es importante validar y sanitizar todas las entradas del usuario, con alguna/s de las siguientes recomendaciones:

* En lugar de intentar eliminar etiquetas específicas (<script>), es más seguro utilizar comandos de programación como *`htmlspecialchars`* para escapar caracteres especiales de HTML, convirtiendo caracteres como `<`, `>`, `&`, `"`, y `'` en entidades HTML, lo que previene la ejecución de código malicioso.
* Inclusión en el código de la cabecera `Content-Security-Policy` (CSP), que permite especificar fuentes de confianza para diferentes tipos de recursos, ayudando a mitigar los ataques de inyección, incluidos los XSS.



* Implementar mecanismos de validación de entrada siempre que sea posible, definiendo un conjunto de reglas (tipos de datos que recibes y características de los mismos) o patrones esperados (uso de expresiones regulares *[regex]* ajustando los datos esperados con las entradas, rechazando cualquier entrada que no se ajuste al patrón), ayudando a prevenir inyecciones maliciosas.

**2.3.4.- Ejemplo de Mejora de código para el nivel medio**



Este código mejorado implementa una política de seguridad de contenido restrictiva que solo permite ejecutar scripts del mismo origen y bloquea 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, protegiendo la aplicación web contra la inyección de HTML/JavaScript

1. RECOMENDACIONES MÁS IMPORTANTES

* Validar y sanear todas las entradas del usuario, asegurando que solo se puedan incluir archivos permitidos.
* Usar antivirus o similar y mantenerlos actualizados.
* Tener todas las aplicaciones actualizadas, especialmente los navegadores 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.
* 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 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.
* 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.

1. CONCLUSIÓNES

Después de analizar los ataques XSS Reflected y las medidas de seguridad para prevenirlos, podemos concluir que la protección contra estos ataques requiere una combinación de prácticas efectivas y una conciencia constante de los riesgos y 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.
* 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.
* El Encoding de salida es el proceso de convertir caracteres especiales en una forma de representación segura en una página web. Por lo que el uso como HTML Encoding, ayudará a prevenir que el navegador interprete el contenido como código malicioso.
* Implementar medidas de educación y concienciación entre los desarrolladores y otros miembros del equipo, para que comprendan los riesgos de XSS Reflected y cómo prevenirlos.
* Realizar revisiones del código fuente regularmente, para identificar cualquier vulnerabilidad XSS Reflected potencial usando herramientas de análisis de código estático y dinámico para ayudar en este proceso.

En resumen, la protección contra ataques XSS Reflected requiere una combinación de prácticas de seguridad efectivas y una conciencia en constante actualización de los riesgos y las vulnerabilidades emergentes. La IA puede ser una herramienta valiosa en la lucha contra los ataques XSS Reflected, ya que pueden ayudar en la automatización de identificación, detección y análisis de patrones maliciosos, ayudando a los analistas de seguridad 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, etc.

En definitiva, es importante recordar que la seguridad es un proceso de mejora continua, que requiere una colaboración constante entre los desarrolladores, los expertos en seguridad y las tecnologías emergentes.

1. 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 ([\](vscode-file://vscode-app/c:/Users/victo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)) y el carácter NULL encontrándose en las versiones nuevas en desuso. [↑](#footnote-ref-1)
2. 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. [↑](#footnote-ref-2)