



INFORME DE VULNERABILIDADES

XSS - Reflected

1.- INTRODUCCIÓN

El presente informe aborda las vulnerabilidades de XSS (Cros Site Scripting) en su tipo “Reflected” 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 – REFLECTED (NIVEL BAJO)

2.1.- Descripción (XSS-Reflected, no persistente o tipo 1):

Es un tipo de ataque web en el que un atacante inyecta código malicioso en una solicitud HTTP que se envía a un servidor web vulnerable, el cual no tiene mecanismos de validación de ese “input”, procesando el código malicioso, que reflejara posteriormente en el navegador del usuario como parte del código HTML de la web, permitiendo al atacante realizar acciones no deseadas por el usuario.

Su funcionamiento se podría dividir en varias etapas:

- ✚ El atacante engaña al usuario para que haga click en el enlace malicioso, normalmente en lenguaje JavaScript, pudiendo estar inyectado en un email, un chat, una red social, etc.
- ✚ El usuario, engañado por el atacante que usa métodos de ingeniería social, se gana la confianza del usuario, el cual, hace click en el enlace del código malicioso, enviando su navegador una solicitud HTTP al servidor web.
- ✚ EL servidor web, carente de suficientes medidas para validar la entrada del usuario, procesa la solicitud del usuario y genera una respuesta HTTP que se incluye en el contenido de la página web.

- ✚ El navegador del usuario recibe la respuesta HTTP del servidor e interpreta el contenido de la pagina web con el código JavaScript malicioso, siendo ejecutado en el navegador del usuario.
- ✚ En este punto, la inyección se ha llevado a cabo y el actor malicioso puede ejecutar acciones no deseadas por el usuario (robo de cookies de inicio de sesión, modificación de contenido de la web, redirigir el usuario a otro sitio web fraudulento e incluso tomar el control del navegador del usuario.

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

```
<?php
header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Feedback for end user
    echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}

?>
```













2.2.1.- Explicación:

1. Apertura del bloque (<?php) y cierre (?>) del script PHP.
2. header ("X-XSS-Protection: 0"); se establece un encabezado HTTP llamado "X-XSS-Protection", siendo una medida de seguridad usada para indicar al navegador el nivel de protección frente ataques XSS, que, en este caso concreto, al asignarle valor cero, le esta indicando que no active ninguna medida de protección. En la actualidad, esta cabecera ha quedado obsoleta salvo para navegadores antiguos¹, debido a las mejoras en los mecanismos internos de los navegadores modernos frente a ataques de XSS junto a las políticas de seguridad de contenido actuales (CSP)² que ofrecen mecanismos mas robustos contra ataques de inyección de código.

```
http-equiv="Content-Security-Policy"
content="default-src 'self'; img-src https://*; child-src 'none';" />
```

¹ <https://developer.mozilla.org/es/docs/Web/HTTP/Headers/X-XSS-Protection>

² <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>

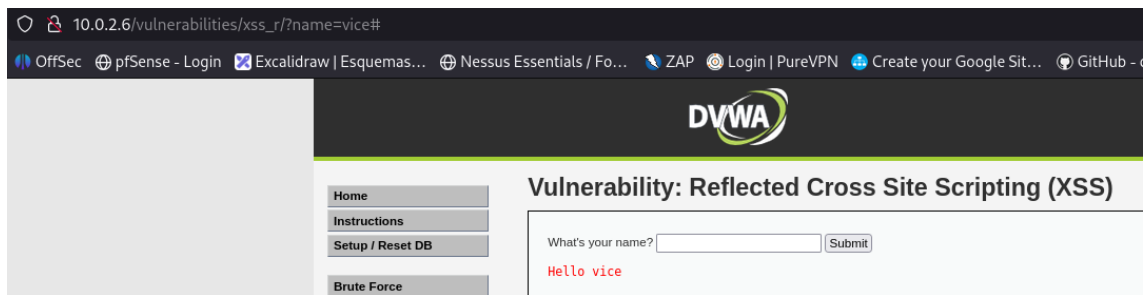
	PC					Móvil					
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
X-XSS-Protection 	 4- 77	 12- 16	 No	 15- 64	 5- 15.3	 18- 77	 No	 14- 55	 4.2- 15.3	 1.0- 11.2	 No

3. `if(array_key_exists("name", $_GET) && $_GET['name'] != NULL) {`
se verifica si existe el parámetro “name” en la solicitud HTTP GET junto a que no sea un valor nulo como condición para pasar a la siguiente línea de código.
4. `echo '<pre>Hello ' . $_GET['name'] . '</pre>';` si se cumplen las condiciones anteriores, se mostrará en código HTML, dentro de una etiqueta “<pre>”, la cadena de texto “Hello”, seguida del valor del parámetro “name”.

El resultado final será una respuesta HTTP que contiene un mensaje de bienvenida personalizado con el nombre del usuario, rodeado por etiquetas <pre> para que se muestre de manera legible.

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.

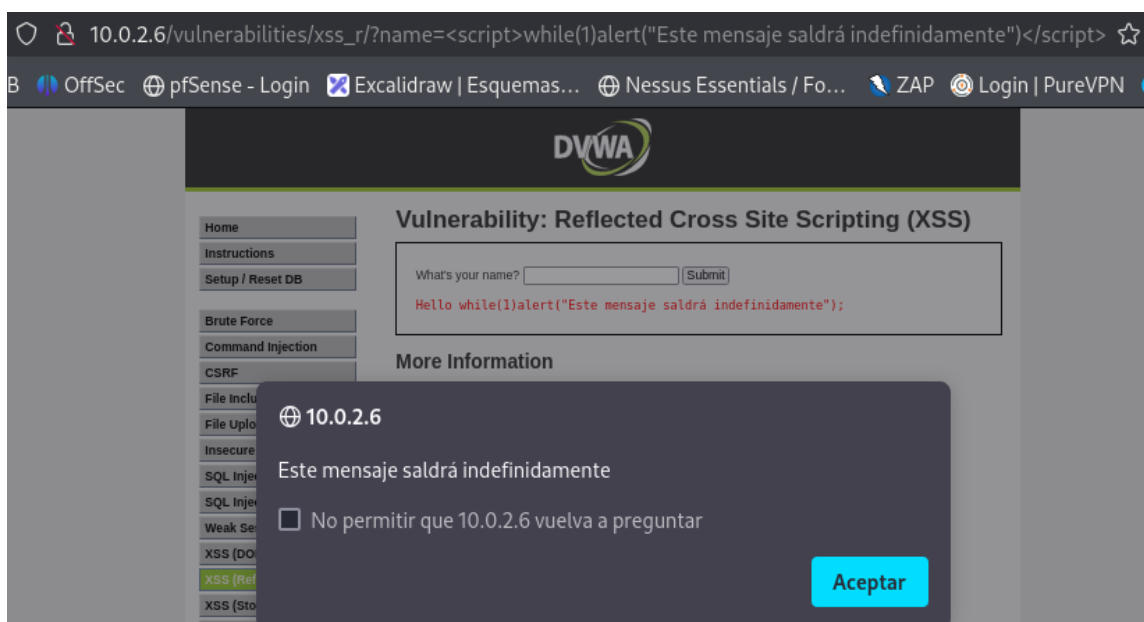


```
<div class="body_padded">
  <h1>
    Vulnerability: Reflected Cross Site Scripting (XSS)
  </h1>
  <div class="vulnerable_code_area">
    <form name="XSS" action="#" method="GET"> *** </form>
    <pre>Hello vice</pre>
  </div>
```

En otro caso, si en la URL detrás del parámetro “name” de la maquina DVWA, ponemos directamente un script, en el que se ejecute este mensaje:

```
<script>while(1)alert("Este mensaje saldrá indefinidamente")</script>
```

Comprobamos que este se ejecutará de manera indefinida, bloqueando la página web con ventanas emergentes cíclicas, hasta que no activemos la casilla “No permitir...”.



En definitiva, se comprueba que este entorno de prueba, es vulnerable a ataques XSS-Reflected. Sin embargo, en un entorno de producción debería haber una fase anterior, que consistiría en ganar la confianza del usuario usando para ello técnicas de ingeniería social.

Por todo ello, es importante asegurarse que cualquier entrada del usuario sea adecuadamente validada y escapada, antes de ser incluida en su salida, para evitar este tipo de vulnerabilidades.

2.2.3.- Mitigación del riesgo

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

1. Client side:

- ✓ Antivirus y aplicaciones instaladas y actualizadas, especialmente el navegador.
- ✓ El uso de software anti – XSS, como XSSAuditor³, que analizan las solicitudes HTTP y elimina algunos JavaScript sospechosos.
- ✓ Eliminación de cookies cada cierto tiempo.

2. Host side:

- ✓ Codificar los datos de requerimientos HTTP HTML en los campos de salida del usuario (Output Encoding)
- ✓ Utilizar frameworks⁴ con buena reputación en seguridad que incluyan los ataques XSS, que automáticamente codifiquen el contenido web para prevenir ataques XSS.
- ✓ Aplicar las “Content Security Policy” (CSP)
- ✓ Implementación de un WAF (Web Application Firewall), que, al igual que en las SQLi, ayudan a impedir la ejecución de ataques XSS.

³ <https://www.chromium.org/developers/design-documents/xss-auditor/>

⁴ conjunto de herramientas predefinidas que proporciona una base para la construcción de aplicaciones web, disminuyendo los intentos de “ensayo-error” para comprobar el funcionamiento del código, permitiendo que estos no se repitan, y concretamente en materia de seguridad, incorporan mecanismos de protección contra las inyecciones de código malicioso.

2.2.4. Ejemplo de Mejora de código para el nivel bajo

```
<?php
// Se sustituye la cabecera de seguridad por la CSP actualizada
header("Content-Security-Policy: default-src 'self'; script-src 'self'; object-src 'none'");

// Verificar si hay alguna entrada y no está vacía
if(array_key_exists("name", $_GET) && $_GET['name'] != NULL) {

    // Escapar caracteres especiales de HTML como "<", ">", "&", "'" y '"'
    // convirtiéndolos en entidades HTML de texto plano para evitar ataques XSS
    $name = htmlspecialchars($_GET['name'], ENT_QUOTES, 'UTF-8');

    // imprime el parametro name junto con un mensaje de bienvenida
    echo "<pre>Hello ' . $name . '</pre>';
}

?>
```

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

```
<?php
header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = str_replace( '<script>', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}

?>
```

2.3.1.- Explicación:

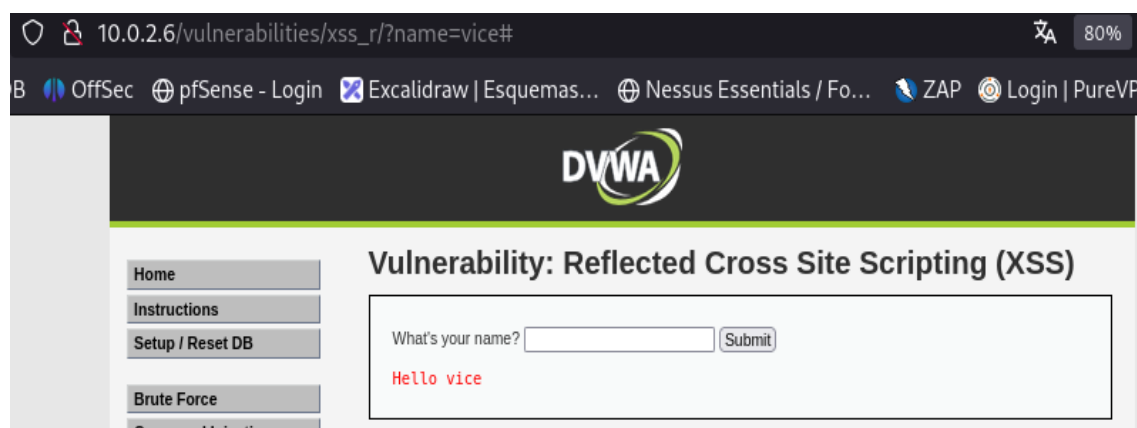
1. Inicio y cierre del documento PHP
2. Al igual que en el nivel bajo se inicia con la cabecera `header ("X-XSS-Protection: 0")`, lo que establece que el filtro de protección contra ataques XSS esta desactivado, pero como ya se ha comentado anteriormente, este método de seguridad ha quedado obsoleto, 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 (``), uso de entidades de HTML que represente a caracteres especiales (`<script>` → `<script>`), o uso de JavaScript en atributos HTML (`Click me`)
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 explicado 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.




```

▼ <div id="main_body">
  ▼ <div class="body_padded">
    ▶ <h1>☒</h1>
    ▼ <div class="vulnerable_code_area">
      ▶ <form name="XSS" action="#" method="GET">☒</form>
      <pre>Hello vice</pre>
    </div>
  </div>

```

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.

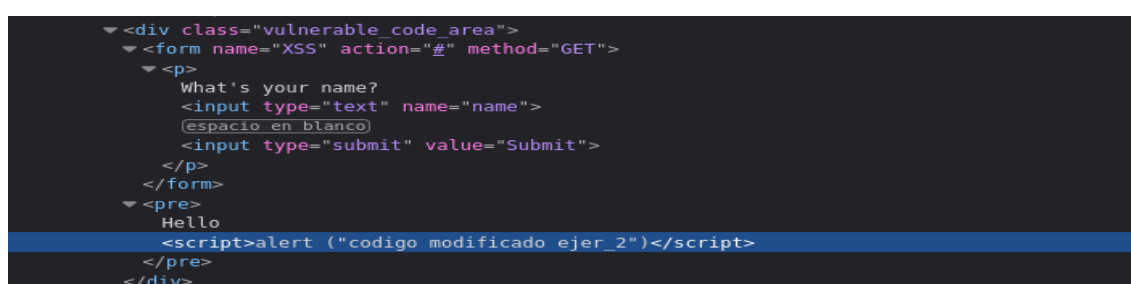
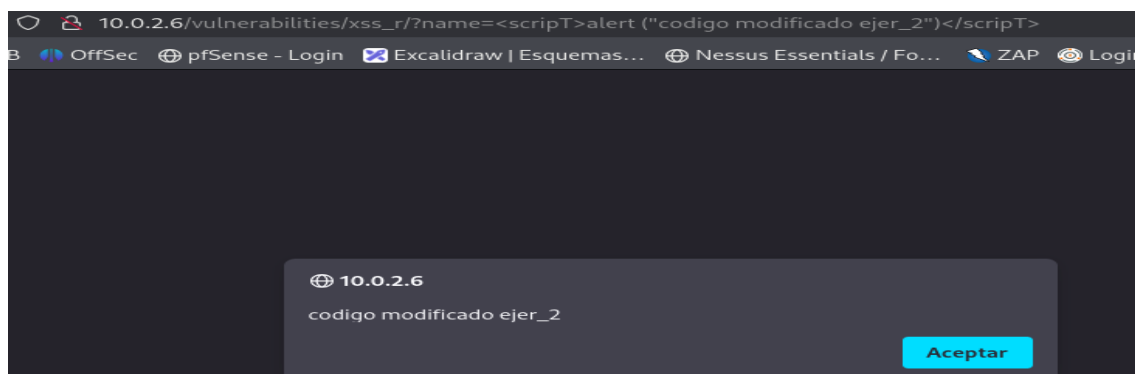


```

▼ <h1>
  Vulnerability: Reflected Cross Site Scripting (XSS)
</h1>
▼ <div class="vulnerable_code_area">
  ▼ <form name="XSS" action="#" method="GET">
    ▼ <p>
      What's your name?
      <input type="text" name="name">
      espacio en blanco
      <input type="submit" value="Submit">
    </p>
  </form>
  <pre>Hello alert("ejercicio nivel medio")</pre>
</div>

```

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.

```
// Se sustituye la cabecera de seguridad por la CSP actualizada
header("Content-Security-Policy: default-src 'self'; script-src 'self'; object-src 'none'");
```

- 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

```
<?php

// Implementar CSP para mejorar la seguridad contra ataques XSS
header("Content-Security-Policy: default-src 'self'; script-src 'none';");

// Verificar si hay alguna entrada y no está vacía
if(array_key_exists("name", $_GET) && !empty($_GET['name'])) {

    // Escapar caracteres especiales de HTML como "<", ">", "&", "'" y '"'
    // convirtiéndolos en entidades HTML de texto plano para evitar ataques XSS
    $name = htmlspecialchars($_GET['name'], ENT_QUOTES, 'UTF-8');

    // imprime el parametro name junto con un mensaje de bienvenida
    echo "<pre>Hello ${name}</pre>";
}

?>
```

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

3. 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 estos 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.

4. CONCLUSIONES

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.