Estudio
Vulnerabilidades persistentes - XSS

Financiado por la Unión Europea
NextGenerationEU

GOBIERNO DE ESPAÑA
MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES

Plan de Recuperación, Transformación y Resiliencia

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD

Financiado por la Unión Europea NextGenerationEU

GOBIERNO DE ESPAÑA MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES

Plan de Recuperación, Transformación y Resiliencia

incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

Marzo 2026

INCIBE-CERT_ESTUDIO_VULNERABILIDADES_PERSISTENTES_XSS_v1.0

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

Índice

ÍNDICE DE FIGURAS Y TABLAS

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

1. Sobre este estudio

La importancia de la gestión eficaz de las vulnerabilidades en el desarrollo de *software* no puede subestimarse en la era digital actual. A medida que las tecnologías evolucionan y se integran en todos los aspectos de la vida cotidiana y la operación empresarial, el espectro de amenazas potenciales se amplía, exponiendo datos críticos o sistemas de información a riesgos significativos. La identificación, evaluación y mitigación oportuna de las vulnerabilidades son fundamentales para salvaguardar la integridad, la confidencialidad y la disponibilidad de los activos digitales.

La persistencia de vulnerabilidades como *Cross-Site Scripting* (XSS), a pesar de ser ampliamente conocida desde los inicios de la navegación web, destaca por la complejidad del desafío al que nos enfrentamos. **¿Por qué sigue vigente hoy en día? Su persistencia se debe a múltiples factores interrelacionados:**

- **Evolución constante de las técnicas de ataque:** los atacantes continúan desarrollando y refinando técnicas para explotar las vulnerabilidades XSS, adaptándose a las nuevas medidas de seguridad y encontrando formas innovadoras de evadir los controles existentes.
- **Complejidad creciente de las aplicaciones web:** a medida que las aplicaciones web se vuelven más complejas e interactivas, aumentan las oportunidades para que los atacantes inyecten *scripts* maliciosos en ellas. La incorporación de múltiples fuentes de datos y la integración con terceros pueden introducir inadvertidamente brechas de seguridad que facilitan los ataques XSS.
- **Dificultades en la validación de los datos de entrada:** a pesar de la conciencia sobre la importancia de la validación de los datos de entrada, la falta de tiempo o recursos puede llevar a implementaciones con una validación insuficiente o incorrecta de los datos introducidos por el usuario.
- **Falta de conciencia y educación:** aunque la conciencia sobre XSS ha aumentado, todavía existe una brecha significativa en la educación y capacitación de desarrolladores y profesionales de la seguridad en cuanto a las mejores prácticas para prevenir y mitigar estos ataques.
- **Legado de código vulnerable:** muchas aplicaciones web antiguas y sistemas o códigos fuente heredados continúan operando sin las actualizaciones necesarias para mitigar las vulnerabilidades XSS, lo que las deja expuestas a ataques.
- **Desafíos en la implementación de medidas de seguridad:** implementar medidas de seguridad efectivas contra XSS, como *Content Security Policy* (CSP), puede ser técnica y organizacionalmente difícil en sistemas grandes y complejos.

Las consecuencias de explotar las vulnerabilidades XSS incluyen accesos no autorizados o robos de identidad que pueden llevar a la exfiltración, modificación, eliminación o inclusión de archivos no deseados, afectando directamente la integridad y confidencialidad de los datos. El riesgo se intensifica cuando estos accesos facilitan el desplazamiento lateral a otros sistemas, extendiendo así el daño más allá del objetivo inicial. Operativamente, la incapacidad para identificar y contrarrestar estas vulnerabilidades de forma proactiva puede impactar negativamente en los procesos críticos del negocio. Por tanto, **es vital reconocer y abordar estas vulnerabilidades en sus fases tempranas**

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

para salvaguardar la integridad de los sistemas de información y mantener la continuidad del negocio.

El presente estudio, dirigido tanto a desarrolladores como a entusiastas tecnológicos, destaca la necesidad de adoptar prácticas de seguridad en aplicaciones web. En él se abordan conceptos fundamentales y se guía a los lectores a través del proceso de configuración y utilización de un laboratorio de pruebas, proporcionando una visión integral de cómo identificar, analizar y mitigar eficazmente los ataques de XSS.

Este enfoque asegura una comprensión completa de la importancia de la seguridad en el desarrollo de aplicaciones web y ofrece herramientas y técnicas esenciales para fortalecer la protección contra amenazas digitales crecientes.

2. Organización del documento

El estudio comienza con la sección **3.- Introducción**, donde se definen los diferentes tipos de XSS, resaltando la necesidad de asegurar las aplicaciones web contra este tipo de vulnerabilidades en un ambiente digital en constante evolución y expuesto a nuevas amenazas.

El apartado **4.- Tipologías de análisis y herramientas** ofrece una visión completa sobre las distintas formas de identificar y analizar vulnerabilidades XSS. Se detallan tanto las metodologías de análisis estático y dinámico como las herramientas disponibles para facilitar la detección de estos fallos de seguridad.

La sección **5.- Preparación de un entorno de test** guía sobre cómo configurar un entorno seguro y controlado para realizar pruebas de vulnerabilidades XSS, esencial para validar la efectividad de las medidas de seguridad implementadas sin comprometer los sistemas de nuestro ecosistema. Posteriormente, en el apartado **6.- Análisis de vulnerabilidades** se presenta un caso de estudio práctico realizado en el entorno anteriormente configurado.

El punto **7.- Remediación y buenas prácticas de desarrollo seguro** se plantean las estrategias claves para mitigar vulnerabilidades XSS, enfatizando en la configuración de políticas de seguridad robustas, programación segura y realización de pruebas y auditorías de seguridad continuas para mantener la integridad, confidencialidad y disponibilidad de los datos.

El documento finaliza con la **8.- Conclusión**, donde se sintetizan los puntos claves y se subraya la importancia de una gestión proactiva e integral de la seguridad en el desarrollo web.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

3. Introducción

La gestión de vulnerabilidades protege a las organizaciones contra ataques y brechas de seguridad y fortalece la confianza de los clientes y usuarios finales en las tecnologías y servicios ofrecidos. Pero dentro del amplio espectro de vulnerabilidades de seguridad, **hay un conjunto persistente de 15 que continúa desafiando a la comunidad de la ciberseguridad a lo largo del tiempo.** Estas vulnerabilidades, que han sido identificadas y clasificadas anualmente en el CWE Top 25 desde 2019 hasta 2023[1], representan fallos críticos que, a pesar de la conciencia generalizada y los esfuerzos de mitigación, siguen siendo prevalentes en el *software* moderno.

Dentro de este ranking, **la vulnerabilidad de *Cross-Site Scripting* (XSS) merece una atención especial. Clasificada consistentemente entre las principales amenazas y situada en el top 2 de vulnerabilidades persistentes en el año 2023, XSS ejemplifica las debilidades asociadas al procesamiento erróneo de datos provenientes de fuentes no confiables**[1], lo que a menudo resulta en un punto de entrada inicial para ataques que comprometen sistemas de TI. Esta vulnerabilidad se caracteriza por su potencial para impactar en numerosas aplicaciones web, brindando a los atacantes la posibilidad de introducir *scripts* maliciosos en sistemas web.

3.1. Definición de XSS

En una aplicación web, el XSS ocurre cuando el sistema no consigue neutralizar o sanitizar de forma segura la información suministrada por los usuarios antes de integrarlos en el contenido de una página web que posteriormente es renderizada por el navegador. Sin un filtrado efectivo que elimine contenido malicioso ejecutable, como JavaScript o HTML, la página generada puede ser comprometida por la ejecución de *scripts* que alteran el comportamiento previsto de la web, poniendo en riesgo la seguridad y privacidad tanto de la aplicación como de sus usuarios. Los ataques XSS se pueden dividir principalmente en tres tipos. **Aunque todas las variantes de XSS (reflejado, almacenado y basado en DOM) se ejecutan en el navegador del usuario, el papel del servidor en cada tipo de ataque varía significativamente**.

XSS Reflejado

El XSS reflejado *(reflected)* es un tipo de ataque en el que el servidor web recibe código malicioso a través de un parámetro en una URL o cualquier otro punto de entrada, como un formulario. Posteriormente, **el servidor procesa esta entrada y devuelve una respuesta al usuario** que incluye el código malicioso que incluyó en la solicitud. Cuando el navegador del usuario interpreta esta respuesta, ejecuta el código malicioso, lo que puede provocar la realización de acciones maliciosas no deseadas. Esto sucede porque el navegador no puede distinguir entre el *script* que es parte legítima de la página y el *script* inyectado por el atacante. También es llamado no persistente porque el *script* malicioso no se almacena en los servidores de la aplicación web, sino que **necesita ser entregado a**

[1] https://cwe.mitre.org/top25/

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

cada víctima de manera individual, a menudo mediante técnicas de engaño que consigan que la víctima visite un enlace malicioso.

Cada uno de estos puntos de entrada representa una potencial vía para que los atacantes inyecten código malicioso. Estos *inputs* incluyen, pero no se limitan a, formularios, campos de texto y directamente la URL, como podemos ver en el siguiente ejemplo.

Imaginemos un escenario en el que un sitio web de búsqueda permite a sus usuarios buscar productos por ID, a través de una URL que incluye un parámetro de búsqueda, como, por ejemplo, `https://www.ejemplo.es/buscar?producto=56`, que utiliza el siguiente código fuente:

```
<?php
// Ejemplo vulnerable a XSS reflejado
// Se obtiene el parámetro 'producto' de la URL mediante GET
$producto = $_GET['producto'];

// El contenido de 'producto' se refleja directamente en la página
echo "Resultados de la búsqueda para: " . $producto;
…
?>
```

Un ataque podría manipular esta funcionalidad abusando del valor de la variable "producto" en una petición HTTP mediante el método GET.

```
https://www.ejemplo.es/buscar?producto=<script>alert("mensaje
arbitrario")</script>
```

Si el usuario hace clic en este enlace, el navegador enviará una solicitud al servidor con la orden de mostrar un mensaje en pantalla. El servidor procesará esta solicitud y devolverá una página que incluye el nombre del producto proporcionado por el usuario. Sin embargo, como el nombre del producto no está siendo correctamente validado o escapado por el servidor, el código JavaScript malicioso dentro de la etiqueta `<script>` será devuelto y ejecutado en el navegador del usuario, mostrando, en este caso, el mensaje deseado.

XSS Almacenado

El XSS almacenado *(stored)*, también conocido como persistente, es un tipo de ataque en el cual **el código malicioso es enviado al servidor web que, a diferencia del XSS reflejado, se almacena en el servidor**. Puede ser introducido a través de formularios, campos de texto o cualquier punto de entrada que permita el almacenamiento de datos en el servidor, como comentarios en un blog, posts en foros o perfiles de usuario. Posteriormente, cuando otros usuarios accedan a una página que incluye los datos almacenados, el navegador interpretará y ejecutará el código especialmente diseñado, afectando a todos los usuarios que visualicen el contenido comprometido. La distinción clave aquí es que el ataque persiste en el servidor y afecta a múltiples usuarios sin necesidad de engañar a cada víctima de forma individual para que visite un enlace malicioso. Los *inputs* vulnerables incluyen, pero no se limitan a, formularios web, áreas de texto y cualquier interfaz que acepte y almacene datos del usuario en el servidor.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

Imaginemos que un usuario envía un comentario a un través de un formulario web como el que se muestra a continuación:

```
<form action="submit_comment.php" method="POST">
    <label for="comentario">Tu comentario:</label>
    <textarea id="comentario" name="comentario"></textarea>
    <button type="submit">Enviar comentario</button>
</form>
```

Y que el archivo PHP submit_comment.php, que procesa la entrada del formulario, incluye este fragmento de código:

```
<?php
// Ejemplo vulnerable a XSS Almacenado
// Guardar comentario
guardarEnBaseDeDatos($_POST['comentario']);
// Se recuperan los comentarios de la base de datos
$comentarios = obtenerComentariosDeLaBaseDeDatos();

// Se muestran los comentarios directamente
foreach ($comentarios as $comentario) {
    echo $comentario;
}
?>
```

Si el atacante incluye en el comentario un código como `<script>alert(“mensaje arbitrario”)</script>`, el *script* malicioso se ejecutará en el navegador de cualquier usuario que vea el comentario, potencialmente comprometiendo su seguridad. En el ejemplo dado provocaría que una alerta apareciera para cada usuario del foro que cargara la página donde se almacene el comentario malicioso. A diferencia del XSS reflejado, el código malicioso reside en el servidor y continúa afectando a los usuarios hasta que es eliminado o neutralizado.

XSS basado en el DOM

El XSS de tipo DOM ocurre cuando se ejecuta código debido a alteraciones en el *Document Object Model* (DOM), resultado de manipulaciones inseguras del entorno por parte del cliente, **sin que este código haya sido alterado o almacenado por el servidor web**. Este tipo de ataque ocurre enteramente en el lado del cliente y puede ser iniciado a través de la manipulación de la URL, formularios web o cualquier interfaz interactiva que altere el DOM basándose en la entrada del usuario. La ejecución del código malicioso se desencadena cuando el JavaScript presente en la página reacciona a estas entradas no sanitizadas, modificando el DOM, de manera que se ejecuten acciones no previstas por los desarrolladores de la página.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

Un ejemplo que ilustra cómo funciona el XSS basado en el DOM es el siguiente. Supongamos que hay una página en un sitio web que permite a los usuarios introducir su nombre en un campo de texto y luego hacer clic en un botón para ver un mensaje de bienvenida personalizado: http://www.ejemplo.es/index.html?usuario=usuarioA, por ejemplo, usando el siguiente fragmento de código:

```
<!-- Ejemplo de un fragmento de código vulnerable a XSS basado en DOM -->
<script>
window.onload = function() {
    // Recupera un valor del parámetro URL 'usuario'
    var usuario = new
URLSearchParams(window.location.search).get('usuario');
    // Inyecta el valor directamente en el DOM
    document.getElementById('mensajeBienvenida').innerHTML = "Bienvenido,
" + usuario;
}
</script>

<div id="mensajeBienvenida"></div>
```

La petición completa, incluyendo la URL con el parámetro usuario, se envía al servidor. Sin embargo, para un ataque XSS basado en DOM, lo crucial no es que el servidor procese o almacene el *script* malicioso, sino cómo el código JavaScript en el cliente (navegador) maneja los datos que recibe de la URL. Cuando el navegador carga la página y ejecuta el *JavaScript* de la misma, este último puede leer los parámetros de la URL directamente desde el cliente sin necesidad de que el servidor intervenga para insertar el *script* en la página. El código JavaScript podría buscar dinámicamente el valor del parámetro usuario en la URL y utilizarlo en el DOM. Al utilizar `innerHTML` como plantea el fragmento de código anterior, el navegador no solo insertaría el texto “Bienvenido, ” en el elemento “mensajeBienvenida”, sino que también procesaría la variable `usuario`. Si esta variable estuviera cargada con `<script>alert(“mensaje arbitrario”)</script>`, `innerHTML` interpretaría su valor como HTML, no como texto plano, permitiendo que el *script* malicioso se ejecutara en el contexto del dominio de la página web y mostrase, en este caso, el mensaje por pantalla.

3.2. Postexplotación del XSS

Las consecuencias postexplotación de un ataque XSS pueden agruparse y ordenarse en las siguientes categorías principales, cada una reflejando distintos niveles de impacto para los usuarios y las organizaciones afectadas:

- **Compromiso de información y sesiones de usuario**: aborda el abuso de la información personal y las sesiones activas de los usuarios, que pueden ser expuestas o secuestradas, permitiendo a los atacantes acceder a cuentas y datos confidenciales sin autorización.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

- **Robo de *cookies*:** los atacantes utilizan *scripts* maliciosos para acceder a `document.cookie` y exfiltrar las *cookies* de sesión del usuario. Dado que las *cookies* a menudo almacenan *tokens* de autenticación, este robo podría permitir al atacante secuestrar sesiones de usuarios, accediendo así a sus cuentas como si fueran ellos mismos.
- **Violaciones de datos:** los atacantes pueden realizar solicitudes (por ejemplo, mediante WebSockets, Fetch API, CORS o incluso Server-Sent Events) en nombre del usuario a API o puntos finales internos que devuelven datos sensibles, permitiéndoles robar información personal, detalles financieros o datos confidenciales de la empresa.
- ***Bypass* de CSRF *Tokens*:** un atacante podría robar los *tokens* CSRF visibles en el DOM y utilizarlos para construir solicitudes maliciosas. Esto le permitiría realizar acciones en la aplicación web en nombre del usuario, como cambiar contraseñas o realizar transacciones, sin que el usuario sea consciente.

- **Manipulación y control del entorno de usuario:** se refiere a la capacidad del atacante para alterar la interfaz o el contenido de la página web que ve el usuario, pudiendo insertar elementos maliciosos o engañosos para realizar *phishing* o distribuir *malware*.
  - **Manipulación de contenido de la página:** insertando *scripts* o HTML malicioso en la página, un atacante puede alterar su contenido para incluir enlaces a sitios de *phishing*, modificar visualmente la información para engañar a los usuarios o insertar formularios falsos con solicitudes de información sensibles, como contraseñas o números de tarjeta de crédito, que envíen datos a servidores controlados por el atacante, como en el caso de los ataques CSRF.
  - **Propagación de *malware*:** un *script* XSS puede redirigir al usuario a una descarga de *malware* o ejecutar un *exploit* directamente en el navegador, buscando instalar *malware* en el dispositivo del usuario para un control remoto, exfiltración de datos o como parte de una *botnet*.
- **Amplificación del ataque y *pivoting*:** muestra cómo los atacantes pueden utilizar un compromiso inicial para escalar o expandir su acceso dentro de la red o sistema objetivo, explotando vulnerabilidades adicionales y comprometiendo más recursos o datos.
  - **Acceso a áreas restringidas y elevación de privilegios:** utilizando la sesión comprometida, el atacante puede intentar acceder a áreas restringidas dentro de la aplicación web, buscando puntos de entrada adicionales o vulnerabilidades que permitan una escalada de privilegios dentro del sistema o la infraestructura.
  - **Exploración de la red interna y cadena de explotación:** con el control inicial obtenido, el atacante puede explorar la red interna de la organización para identificar otros sistemas vulnerables, utilizando técnicas como la falsificación de peticiones (SSRF) para expandir el alcance del ataque más allá de la aplicación web inicialmente comprometida.

4. Tipologías de análisis y herramientas

Existen diferentes enfoques para detectar y analizar las vulnerabilidades XSS, incluyendo análisis estático, dinámico y en tiempo de ejecución.

En el SAST o **análisis estático** se examina el código de la aplicación sin ejecutarlo. Las herramientas de análisis estático buscan patrones de código conocidos que podrían indicar la presencia de vulnerabilidades XSS. Herramientas como SonarQube o Brakeman (para aplicaciones Ruby on Rails) son ejemplos destacados en este ámbito que se enfocan en lugares donde el código no emplea métodos de escape o *encoding* correctamente para datos de usuario, lo cual debería incluir no solo URL *encoding*, sino también HTML, JavaScript, y CSS *encoding* dependiendo del contexto de uso. Por ejemplo, SonarQube revisa si se utiliza `innerHTML` o `document.write` con variables sin sanitizar y si se aplican funciones específicas de *encoding*, como `htmlEscape(userInput)` para HTML, `encodeURIComponent(userInput)` para URL, y métodos similares para JavaScript y CSS, evitando así que caracteres peligrosos se interpreten como parte del código ejecutable.

El DAST o **análisis dinámico** conlleva interactuar con la aplicación en ejecución para identificar vulnerabilidades de seguridad, incluidas las de tipo XSS. Utiliza técnicas como inyecciones de datos maliciosos para ver cómo responde la aplicación, intentando explotar posibles vulnerabilidades. OWASP ZAP[2] y Burp Suite[3] (Community Edition) son ejemplos de herramientas que sirven para el análisis dinámico de aplicaciones. Ambas funcionan como *proxies* intermediarios, permitiendo interceptar y manipular el tráfico entre el navegador y el servidor web para analizar y modificar solicitudes y respuestas HTTP/HTTPS. Durante el análisis dinámico, inyectan *payloads* que prueban una variedad de técnicas de *encoding*, incluyendo URL, HTML y JavaScript *encoding*, para evadir filtros de seguridad y detectar vulnerabilidades XSS. ZAP también prueba las versiones de inyección HTML *encoded*, JavaScript *encoded* y otras variantes para comprobar cómo la aplicación maneja diferentes *encodings*.

El **análisis en tiempo de ejecución** implica monitorizar la aplicación mientras se ejecuta en un entorno de producción o similar, identificando o mitigando vulnerabilidades XSS en tiempo real. Esta técnica a menudo utiliza instrumentación de aplicaciones y herramientas de protección en tiempo de ejecución (RASP) para detectar y bloquear ataques XSS mientras ocurren. Aunque no hay muchas plataformas abiertas dedicadas exclusivamente a la protección RASP, referencias como OWASP AppSensor[4] pueden ayudar a implementar mecanismos de detección y respuesta en tiempo real dentro de las aplicaciones web. AppSensor puede ayudar a los desarrolladores a diseñar e integrar la lógica de seguridad que responde activamente a eventos anómalos o maliciosos de tipo XSS detectados durante la ejecución de la aplicación. Por otro lado, los componentes WAP, como modsecurity[5] u OWASP Core Rule Set (CRS)[6], pueden ayudar a monitorizar el tráfico HTTP sospechoso y bloquearlo. Por ejemplo, implementa reglas que detectan

[2] https://www.zaproxy.org/

[3] https://portswigger.net/burp

[4] https://owasp.org/www-project-appsensor/

[5] https://www.modsecurity.org

[6] https://owasp.org/www-project-modsecurity-core-rule-set/

ataques XSS, no solo mediante patrones directos, sino también a través de diversas técnicas de *encoding*, como HTML, URL, Base64 y JavaScript *encoding*. Esta capacidad para decodificar y analizar múltiples formatos de *encoding* es esencial para identificar y bloquear intentos de inyección de *scripts* maliciosos que usan *encodings* complejos para ocultar sus cargas.

5. Preparación de un entorno de test

Para establecer un entorno seguro destinado a la prueba y análisis de vulnerabilidades XSS, resulta crucial disponer de un laboratorio de seguridad aislado. Este laboratorio permitirá realizar pruebas en un entorno controlado, minimizando los riesgos para los sistemas de producción y ofreciendo la flexibilidad necesaria para simular diversos escenarios de ataque.

Una forma efectiva de configurar dicho laboratorio es mediante el uso de máquinas virtuales que faciliten la creación de entornos aislados y replicables. VirtualBox[7], una herramienta gratuita de virtualización, se presenta como una opción ideal para este propósito gracias a su facilidad de uso y a la amplia disponibilidad de recursos y documentación.

Para el entorno de pruebas que será vulnerable a XSS, existen varios recursos disponibles gratuitos en Internet. Uno de ellos es la máquina DVWA[8] (Damn Vulnerable Web Application), ampliamente reconocida y utilizada en el ámbito del *pentesting* web por proporcionar un servidor Linux con una aplicación web con una serie de vulnerabilidades predefinidas, incluyendo, pero no limitándose a, XSS. Este tipo de aplicaciones vulnerables son ideales para entrenamientos en seguridad informática, ya que permiten a los usuarios experimentar con ataques en un entorno seguro y aprender sobre las medidas de protección adecuadas.

El paso inicial para configurar este entorno de laboratorio incluye crear una nueva máquina virtual dentro de VirtualBox a partir de la imagen ISO de DVWA.

[7] https://www.virtualbox.org/

[8] https://www.vulnhub.com/entry/damn-vulnerable-web-application-dvwa-107,43/

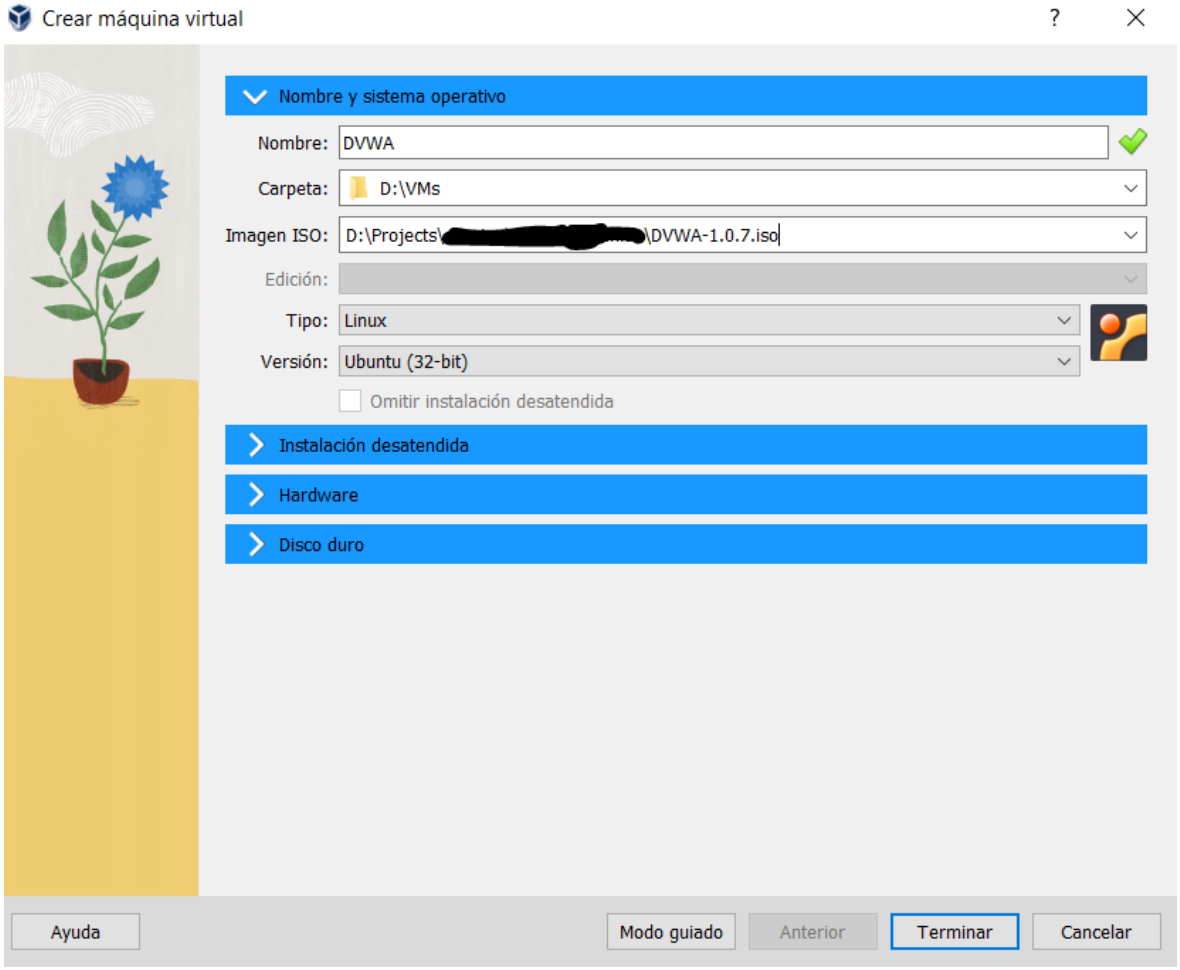


Ilustración 1: Creación máquina vulnerable

Tras la creación de la máquina virtual, es esencial desplegarla en un segmento de red que esté aislado, evitando así cualquier acceso externo desde fuera del dispositivo donde se ejecuta. Una forma práctica de lograr esto es ajustando la interfaz de red de la máquina virtual a modo NAT. En VirtualBox, este modo asigna a la máquina virtual una dirección IP de una red privada gestionada por el propio *software* de virtualización, accesible únicamente desde el dispositivo anfitrión. Este método de configuración es similar al modo "solo anfitrión", con la ventaja de permitir la inclusión de más máquinas virtuales en dicha red, si fuera preciso aumentar el laboratorio. Para configurar esto, se debe seleccionar la máquina virtual en la interfaz de VirtualBox, dirigirse a la sección de red y modificar la configuración del adaptador conforme se indica en la imagen adjunta.

*Ilustración 2: **Configuración adaptador de red***

Para terminar de realizar el despliegue del entorno solo queda desplegar DVWA. Se ha elegido el modo de instalación principalmente por sus ventajas adicionales, como la posibilidad de personalizar y configurar el entorno de manera más detallada, lo que puede ser útil para propósitos específicos de aprendizaje o prueba. No obstante, es importante destacar que el modo *live* también ofrece una opción viable para la restauración del sistema, gracias a la funcionalidad de capturas de estado *(snapshots)* proporcionada por el *software* de virtualización. Estas capturas permiten guardar el estado completo de la máquina virtual en un momento dado, facilitando la restauración a ese punto específico si es necesario, sin importar si la máquina se encuentra en modo instalación o *live*.

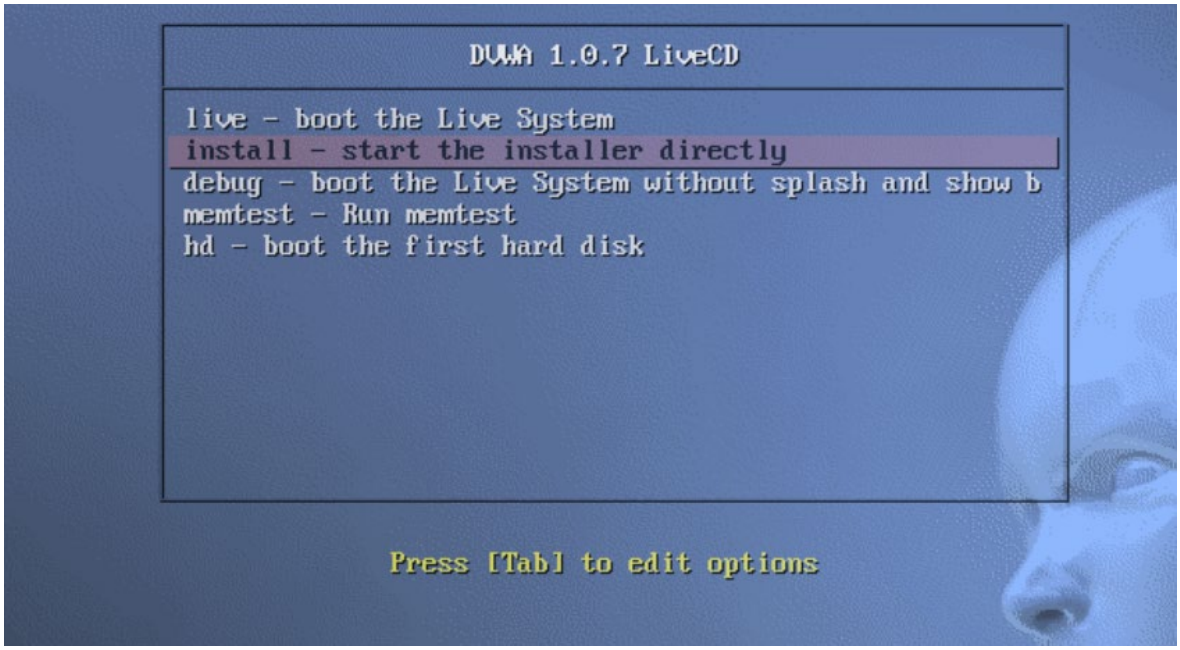


*Ilustración 3: **Instalación DVWA***

En cuanto termine este proceso, el entorno estará listo, las credenciales por defecto de la máquina DVWA son: *admin* / *password*. Con el comando `ifconfig` comprobamos la dirección IP que ha sido asignada a la máquina. Con esta información, es posible acceder a la aplicación web mediante un navegador, dirigiéndose a la URL "http://<IPDVWA>/". Este enlace redireccionará automáticamente a "http://<IPDVWA>/login.php", donde será necesario emplear las credenciales mencionadas anteriormente para iniciar sesión.

Tras acceder al sistema, es importante configurar el nivel de seguridad del entorno. La aplicación brinda la posibilidad de ajustar diversos niveles, los cuales modifican directamente la rigurosidad de las medidas de seguridad implementadas. Por ejemplo, al seleccionar un nivel de seguridad más bajo, se desactivan distintos mecanismos de control sobre los datos introducidos por los usuarios. Para nuestra prueba lo ideal es asignar un nivel bajo a través de la opción "DVWA Security" en el menú. La manera de ajustar esta configuración se ilustra en la siguiente imagen:

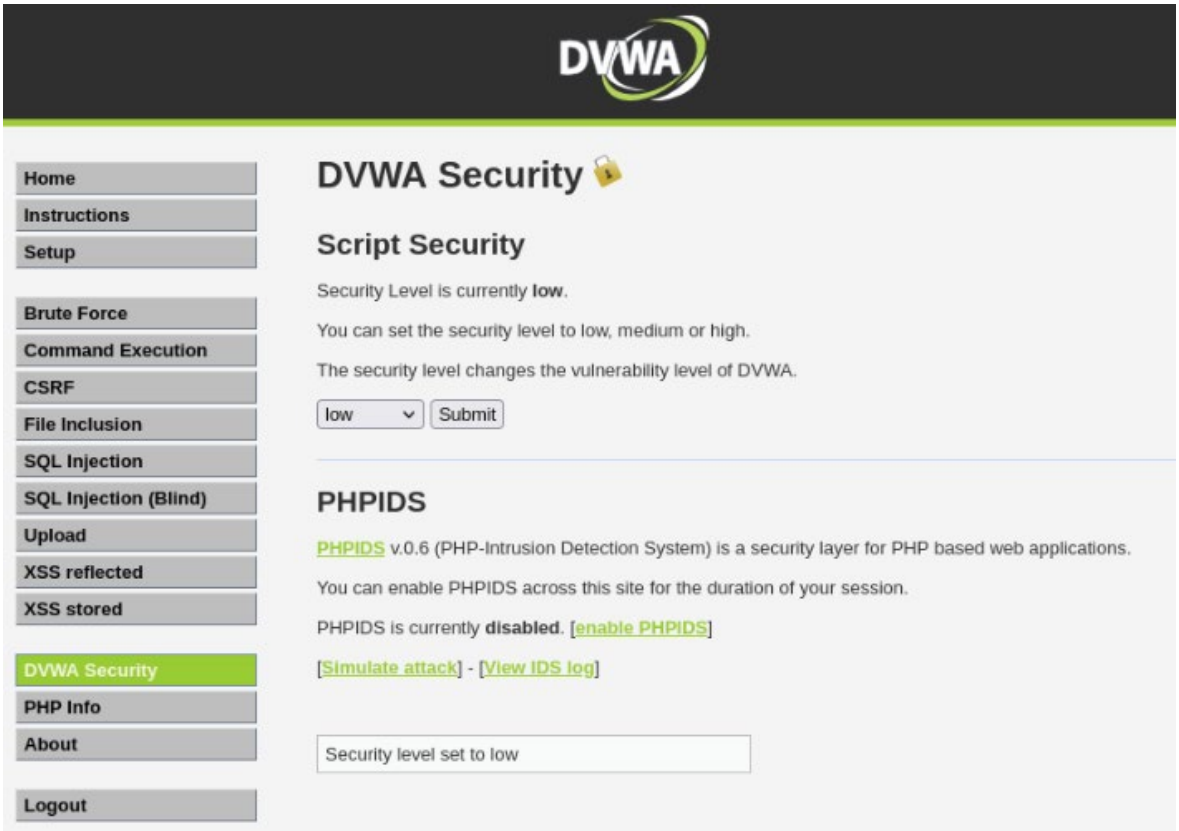


Ilustración 4: Configuración nivel de seguridad DVWA

Con esto ya estaría el entorno de pruebas listo para interactuar con él y poder estudiar las vulnerabilidades XSS.

6. Ejemplos de análisis de vulnerabilidades XSS

En nuestro caso práctico, realizaremos el análisis de dos vulnerabilidades: un XSS reflejado y otro almacenado.

Optaremos por realizar pruebas de tipo DAST, las cuales son fundamentales para identificar vulnerabilidades en sistemas web en tiempo de ejecución. Para esta guía, hemos seleccionado OWASP ZAP como la herramienta de prueba, aunque Burp Suite hubiera sido igualmente adecuada para el propósito. Ambas herramientas están disponibles por defecto en todas las distribuciones de Kali Linux, lo que facilita su acceso y uso.

Para encontrar ZAP en Kali Linux, simplemente se puede utilizar el buscador en el menú de aplicaciones escribiendo el término “ZAP”. Al ejecutar la aplicación, se abrirá el menú de inicio rápido, que proporciona un acceso directo a las funciones más importantes de la herramienta, facilitando el inicio de las pruebas de seguridad de manera inmediata.

Ilustración 5: Índice de OWASP ZAP

De las opciones que muestra la aplicación, primero se va a seleccionar la opción “Escaneo automático”, que mostrará una pantalla que solicita la URL a atacar y el navegador a utilizar para realizar el proceso de *spidering*. Este proceso permite a la herramienta descubrir y mapear de manera automática los recursos disponibles en una aplicación web, identificando los puntos de entrada que serán luego analizados en busca de vulnerabilidades.

Ilustración 6: Modo escaneo automático

Al pulsar en el botón iniciar ataque, la herramienta empezará a realizar ciertas operaciones para comprobar algunas vulnerabilidades web. En el panel de alertas se podrán ir viendo las detecciones que recoge la herramienta. Entre estas alertas ya aparece alguna información que da a entender que esta aplicación podría ser vulnerable a XSS. En la imagen que se muestra a continuación se pueden observar algunas de las alertas detectadas por la herramienta:

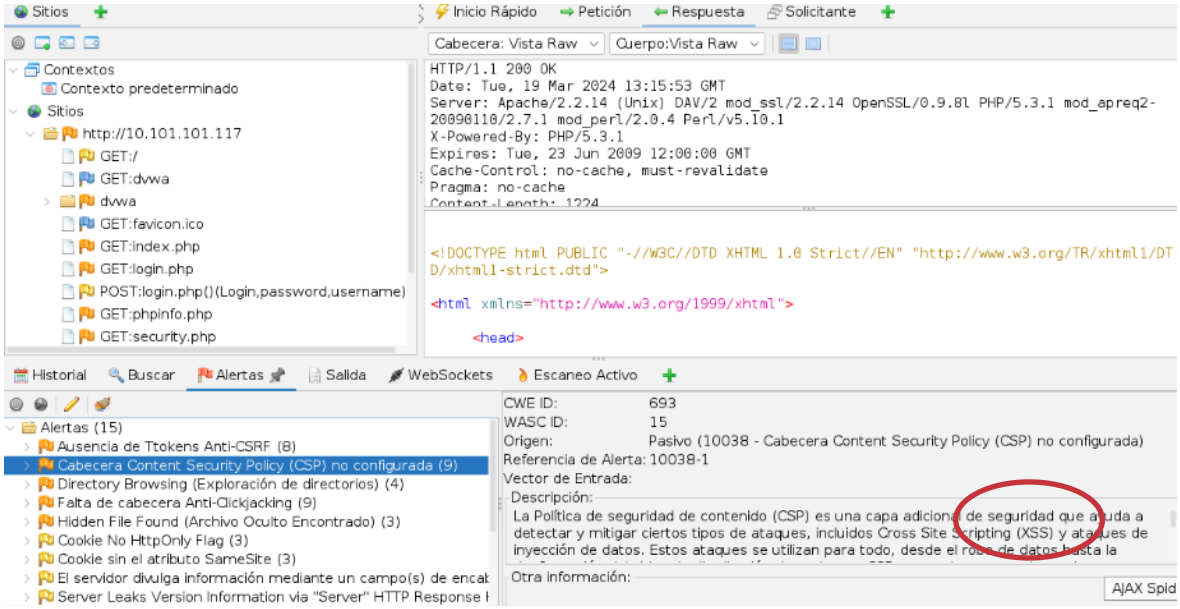


Ilustración 7: Alertas de hallazgos

Como se aprecia en la imagen anterior, la herramienta detecta que faltan varias cabeceras de seguridad que precisamente protegen frente al XSS. El siguiente paso es comenzar con la exploración manual. Para esto, en la pestaña inicio rápido, se debe seleccionar la opción de exploración manual, en la cual se pide la URL a la que se va a atacar y el navegador con el que se quiere realizar la exploración manual.

Ilustración 8: Modo de escaneo manual

XSS reflejado

Usando las credenciales comentadas en la sección 5, se puede realizar *login* en la aplicación y comenzar con el ejercicio de explotación de la vulnerabilidad XSS reflejada. Para esto, se debe navegar hacia la sección "XSS Reflected", donde la aplicación muestra la siguiente pantalla:

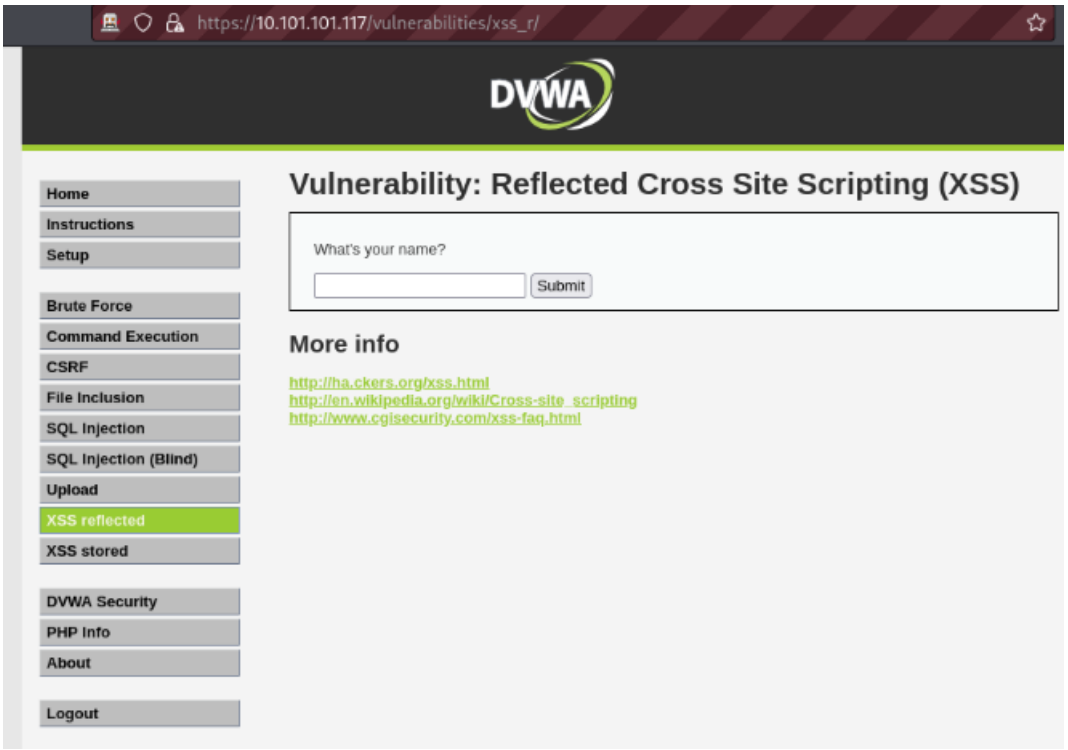


Ilustración 9: Panel de funcionalidad "XSS Reflected"

Se muestra una caja de texto acompañada de un botón de *submit*. Para ver en detalle qué hacen las acciones de esta aplicación web, se necesita activar el *proxy* de ZAP para que empiece a parar peticiones. En ZAP esto se llama punto de interrupción y se activa

haciendo clic en la esfera verde que se puede encontrar en la barra de herramientas de ZAP. Cuando ZAP intercepta la petición se captura la información que se envía al servidor antes de que se produzca el envío.

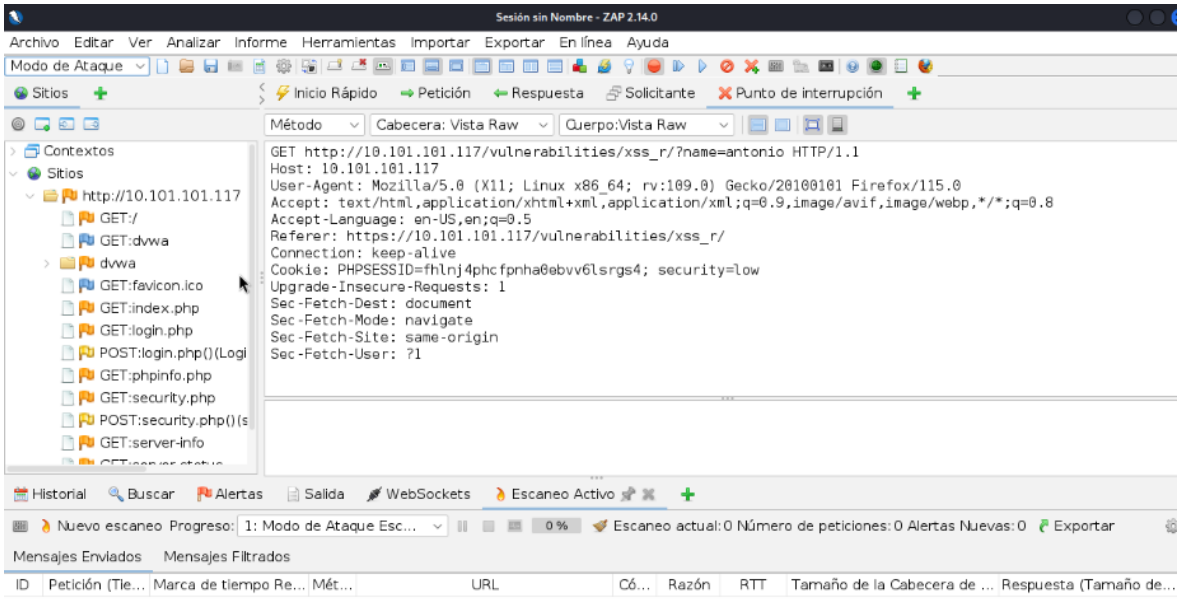


Ilustración 10: Petición de funcionalidad "XSS Reflected"

Podemos apreciar cómo el valor introducido en la caja de texto viaja como parámetro "name" en la URL. Para realizar un ataque, como se puede observar que el valor de "name" se renderiza después de ser enviado y se muestra en la página resultado, se va a cambiar el valor de la variable por una carga maliciosa. Como se ha explicado previamente, la intención de este ataque es ganar la capacidad de cargar y ejecutar un *script* en el contenido que devuelve la web. Por lo tanto, cambiamos el valor introducido previamente por "`<script>alert('XSS');</script>`". Esta modificación de la petición se puede apreciar en la siguiente imagen:

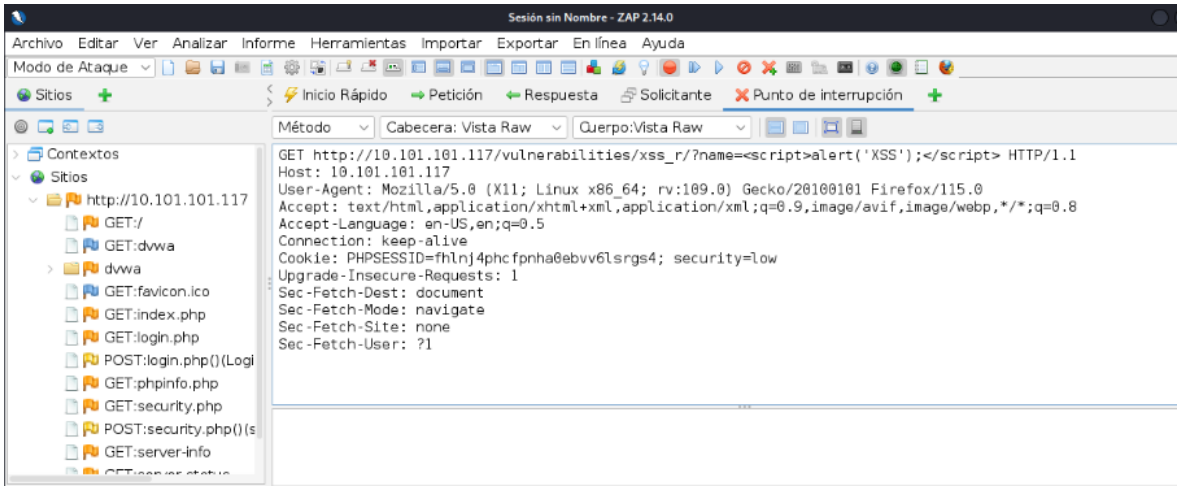


Ilustración 11: Modificación de petición de funcionalidad "XSS Reflected" antes de ser enviada al servidor

Al realizar el cambio en el parámetro “name”, se deja pasar la petición haciendo clic en el botón que hay justo a la derecha del punto de interrupción, y al volver al navegador, se puede apreciar cómo el resultado del ataque ha sido exitoso y se ha conseguido ejecutar el código.

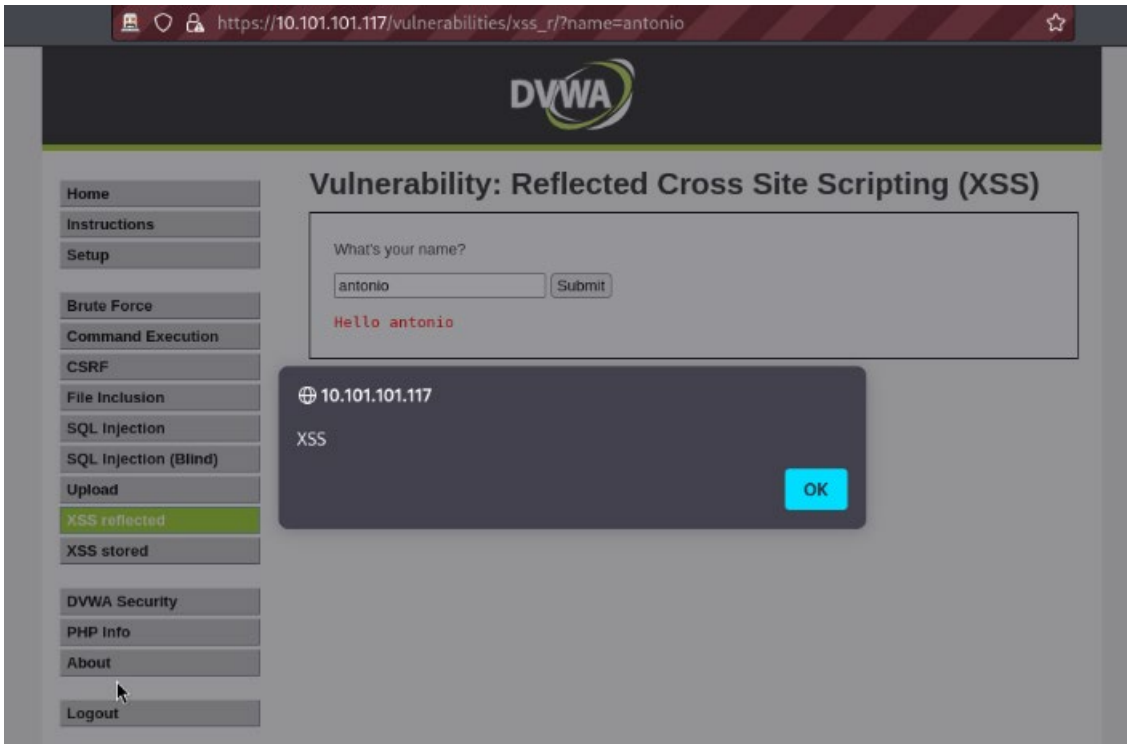


Ilustración 12: Explotación de XSS reflejado

Esta situación ocurre porque el valor del parámetro “name” se refleja en la respuesta del servidor tal cual se recibe, sin realizar una sanitización o procesamiento previo. Esto permite que el usuario suministre código a través de este parámetro que, una vez renderizado por el navegador, puede resultar en un comportamiento inesperado o malicioso de la aplicación.

En la siguiente imagen se aprecia el código fuente de la funcionalidad con la que se está interactuando. En este fragmento de código se aprecia cómo no existe un procesamiento de la entrada de datos:

```
<?php

header ("X-XSS-Protection: 0");

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

?>
```

Ilustración 13: Código vulnerable de la funcionalidad “XSS Reflected”

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

La solución a este problema radica en asegurarse de que los datos no controlados, es decir, aquellos proporcionados por el usuario, no se rendericen directamente en la página web **sin antes ser debidamente procesados**. La práctica más común y efectiva para prevenir este tipo de vulnerabilidades es la **sanitización de la entrada de datos**. La clave es la implementación de una función de sanitización, como `htmlspecialchars`, un método estándar en este proceso como veremos en la sección 7. Esta función convierte caracteres especiales en entidades HTML, que previenen la ejecución de código malicioso inyectado en la aplicación. Por ejemplo, si un usuario intenta insertar un *script* mediante la entrada de datos, `htmlspecialchars` transformará los caracteres relevantes (<, >, ", ', &) en sus equivalentes de entidad HTML (<, >, ", ', &), evitando que el navegador interprete el *input* como código ejecutable.

```php
<?php

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
        // Check Anti-CSRF token
        checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

        // Get input
        $name = htmlspecialchars( $_GET[ 'name' ] );

        // Feedback for end user
        $html .= "<pre>Hello {$name}</pre>";
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

Ilustración 14: Código con las correcciones aplicadas sobre "XSS Reflected"

XSS almacenado

Para comenzar con la explotación del XSS almacenado se debe ir a la sección del menú de la aplicación "XSS Stored". En esta parte de la aplicación se presentan dos campos de entrada de texto que permiten al usuario introducir un nombre de usuario y un texto para publicar un comentario en la página web.

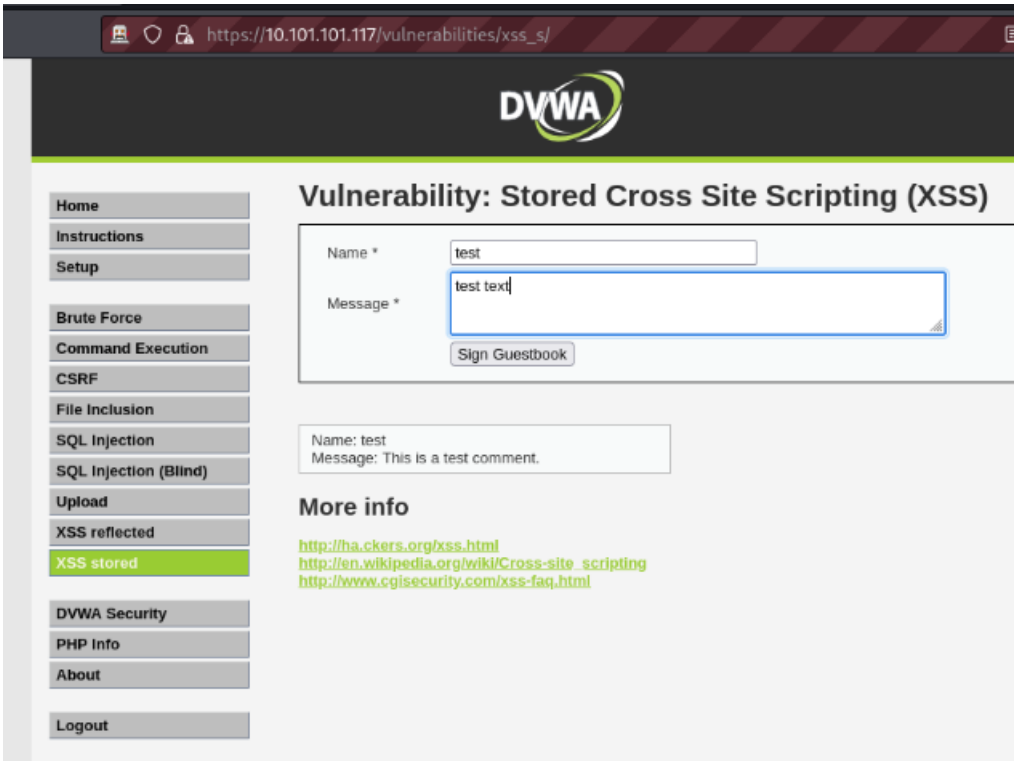


Ilustración 15: Panel de funcionalidad "XSS Stored".

Tal como se vio en el caso anterior, se puede realizar un punto de interrupción en ZAP para poder ver qué información y cómo se envía ésta al servidor. Tal y como se muestra en la siguiente imagen, la petición usa el método POST para enviar información en la parte de datos de la petición:

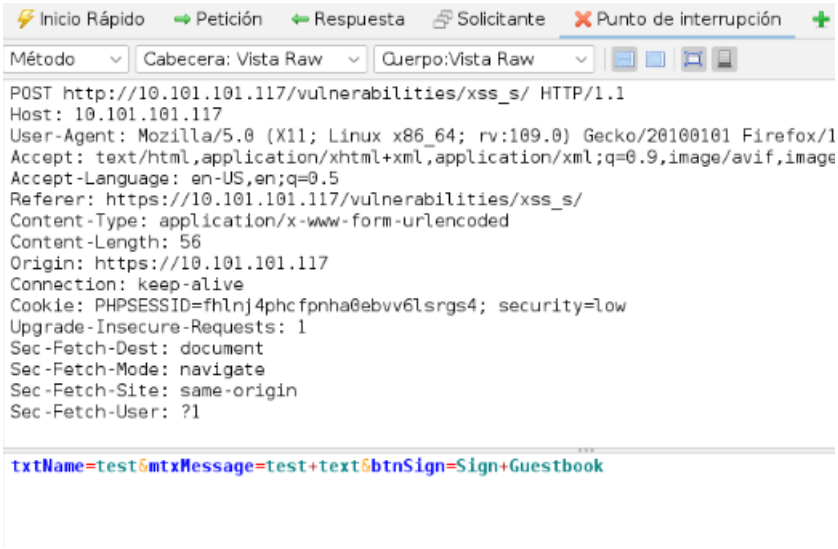


Ilustración 16: Petición de la funcionalidad "XSS Stored"

Al igual que para el ejemplo anterior, se debe sustituir alguna de las partes de la carga por un *payload* malicioso que altere el comportamiento de la web cuando intente renderizar ese contenido. En la siguiente imagen se aprecia cómo se modifican los campos de la petición recibida incluyendo en uno de ellos la carga maliciosa:

Ilustración 17: Modificación de petición de funcionalidad "XSS Stored" antes de ser enviada al servidor

Cuando, como cliente, se solicita el acceso de nuevo a la sección "*XSS Stored*" y el navegador renderiza el contenido de los comentarios, se aprecia cómo se muestra la alerta XSS, lo que demuestra el comportamiento alterado en base a la carga útil introducida.

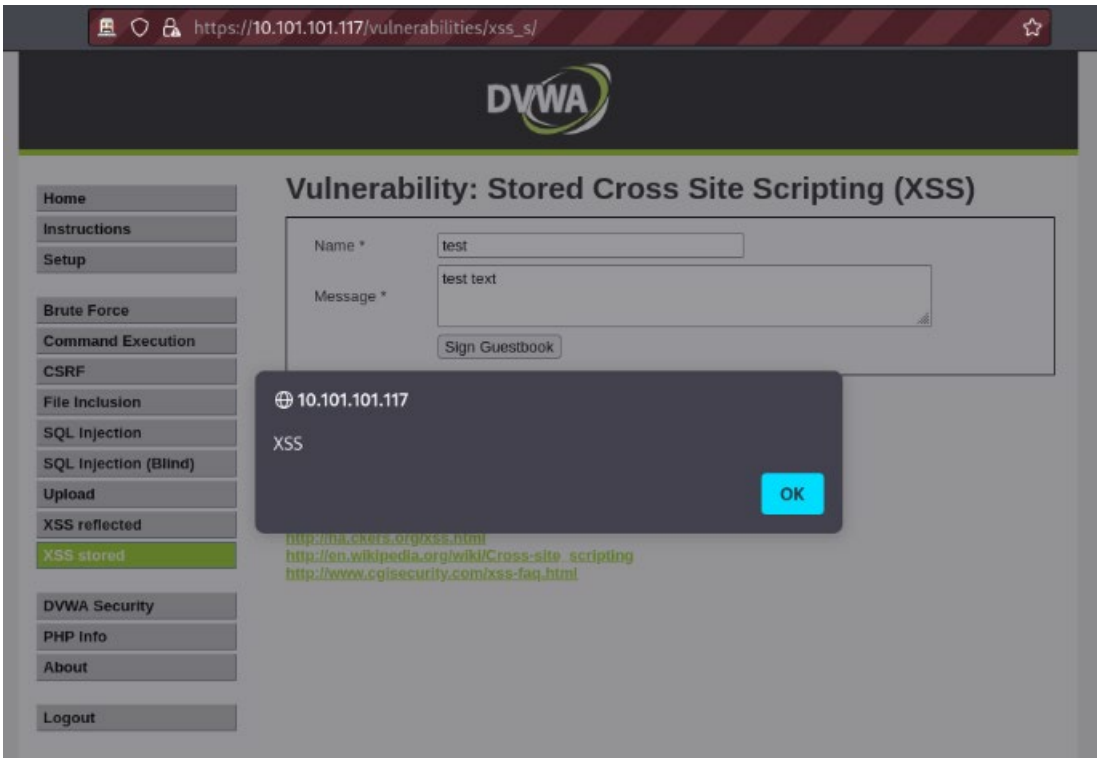


Ilustración 18: Explotación de XSS almacenado

Este problema de seguridad se da porque solo se controla el nombre de usuario a la hora de sanitizar la entrada de usuario, por lo que la parte del mensaje no se valida correctamente, permitiendo ingresar la carga útil maliciosa. Eso se puede apreciar en la siguiente imagen que muestra el código fuente vulnerable de esta funcionalidad:

```
<?php

if( isset( $_POST['btnSign'] ) ) {
    // Get input
    $message = trim( $_POST['mtxMessage'] );
    $name = trim( $_POST['txtName'] );

    // Sanitize message input
    $message = stripslashes( $message );

    $message = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message ) : ((trigger_error("[MySQLConverterToo] Fix the
    mysqli_escape_string() call! This code does not work.", E_USER_ERROR)));

    // Sanitize name input
    $name = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name ) : ((trigger_error("[MySQLConverterToo] Fix the
    mysqli_escape_string() call! This code does not work.", E_USER_ERROR)));

    // Update database
    $query = "INSERT INTO guestbook ( comment, name ) VALUES ( '$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>' );

    mysql_close();
}

?>
```

Ilustración 19: Código fuente vulnerable de la funcionalidad "XSS Stored"

La clave para asegurar la seguridad de la aplicación web reside en **monitorizar y validar cuidadosamente todas las áreas de entrada de datos** con las que el usuario puede interactuar. El uso de `htmlspecialchars`, como en el caso anterior, ayudaría a mitigar este riesgo. Este proceso de sanitización es esencial para eliminar o neutralizar cualquier contenido introducido por el usuario que pudiera afectar negativamente el funcionamiento normal de la aplicación. La imagen que se presenta a continuación ilustra cómo se revisan y limpian todas las entradas de los campos de mensaje, garantizando así que solo se procesen datos seguros y limpios en la aplicación:

```
<?php

if (isset($_POST['btnSign'])) {
    // Check Anti-CSRF token
    checkToken($_REQUEST['user_token'], $_SESSION['session_token'], 'index.php');

    // Get input
    $message = trim($_POST['mtxMessage']);
    $name = trim($_POST['txtName']);

    // Sanitize message input
    $message = stripslashes($message);
    $message = ((isset($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)) ? "" : ""));
    $message = htmlspecialchars($message);

    // Sanitize name input
    $name = stripslashes($name);
    $name = ((isset($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)) ? "" : ""));
    $name = htmlspecialchars($name);

    // Update database
    $data = $db->prepare('INSERT INTO guestbook (comment, name) VALUES (:message, :name)');
    $data->bindParam(':message', $message, PDO::PARAM_STR);
    $data->bindParam(':name', $name, PDO::PARAM_STR);
    $data->execute();

    // Generate Anti-CSRF token
    generateSessionToken();
}

?>
```

Ilustración 20: Código con las correcciones aplicadas sobre "XSS Stored".

7. Prevención y buenas prácticas

Prevenir y solucionar las vulnerabilidades XSS implica una serie de estrategias y prácticas de codificación segura para proteger las aplicaciones web contra la inserción y ejecución de *scripts* maliciosos. Aquí se detallan algunas acciones claves para la prevención y remediación de XSS.

7.1. Configuración de políticas seguras

La política de mismo origen (SOP) es una medida de seguridad esencial en los navegadores que restringe la interacción entre documentos o *scripts* de diferentes orígenes para proteger la información del usuario contra ataques de tipo XSS. CORS refleja de forma controlada esta política, permitiendo accesos cruzados específicos mediante cabeceras HTTP, facilitando así la compartición segura de recursos web entre dominios distintos. Por otro lado, la política de seguridad de contenido (CSP) ofrece una capa de seguridad adicional que permite a los desarrolladores definir qué recursos pueden cargarse en sus páginas web, ayudando a prevenir la ejecución de *scripts* maliciosos mediante la especificación de fuentes de confianza para *scripts*, estilos y otros recursos. Y por supuesto, la programación o codificación segura.

En la aplicación web y el servidor

La implementación de CORS y CSP recae en los diseñadores, arquitectos y desarrolladores de la aplicación, quienes juegan un papel crucial en la prevención de la ejecución de *scripts* no autorizados. Para lograr este objetivo, existen varias medidas fundamentales que los desarrolladores deben adoptar respecto a estos mecanismos de seguridad:

- **Uso responsable de CORS:** como regla general, se recomienda garantizar que las solicitudes de AJAX, Fetch API y otros métodos de solicitud de recursos estén restringidos a llamadas de mismo origen, salvo excepciones. Para operaciones sensibles (como acciones POST que cambian el estado), se debe verificar el origen de la solicitud para asegurarse de que coincida con el origen esperado y solo se debe utilizar CORS cuando se necesite compartir recursos entre orígenes diferentes, especificando exactamente qué orígenes tienen permitido acceder a los recursos y qué métodos HTTP pueden utilizar. Es importante evitar configuraciones demasiado permisivas que puedan ser explotadas por atacantes. Adicionalmente, se puede considerar el aislamiento de partes sensibles de la aplicación web (como la administración o el manejo de datos personales) en diferentes orígenes (subdominios) para limitar aún más el alcance de los posibles ataques XSS.
- **Implementación correcta de CSP:** se recomienda iniciar con un análisis detallado del inventario de recursos empleados por la aplicación web, identificando claramente sus orígenes. Se puede elaborar una política preliminar que limite la carga de recursos únicamente a aquellos orígenes verificados y seguros, aplicando directivas específicas (como `script-src`, `style-src`, `img-src`) para restringir distintos tipos de contenido a fuentes confiables. Esta política puede ser implementada inicialmente en modo “report-only”, a través de la cabecera “Content-Security-Policy-Report-Only”, permitiendo la recolección de informes de violaciones sin afectar la operatividad de la web. La recopilación y análisis de estos informes

facilita el ajuste preciso de la política, incluyendo la posibilidad de incorporar *hashes* o *nonces* para *scripts* y estilos *inline* específicos, garantizando su validez sin abrir excepciones generales. Posteriormente, tras una fase de ajuste y validación, la política se puede aplicar de manera definitiva mediante la cabecera *Content-Security-Policy*, reemplazando el modo `report-only` y asegurando una monitorización continua para adaptaciones futuras.

En el navegador

SOP es una medida de seguridad implementada por los navegadores web. Para no depender de SOP, si un desarrollador está preocupado por la seguridad y el posible *bypass* de SOP (a través de vulnerabilidades del navegador, configuraciones de usuario inseguras o extensiones de navegador maliciosas), la estrategia más efectiva es implementar medidas de seguridad adicionales en el lado del servidor y en la aplicación web.

7.2. Programación segura

La programación segura es una estrategia fundamental para limitar los efectos y la viabilidad de los ataques de XSS. Adoptar prácticas de codificación segura minimiza las vulnerabilidades en las aplicaciones web que los atacantes podrían explotar para inyectar y ejecutar *scripts* maliciosos. Algunas de las técnicas claves de programación segura para prevenir XSS incluyen:

- **Validación de las entradas:** es importante asegurarse de que todas las entradas recibidas de los usuarios sean validadas. Se pueden utilizar listas blancas para permitir solo los caracteres o patrones esperados y rechazar cualquier entrada que no cumpla con estos criterios. La sanitización de las entradas de usuario elimina o modifica caracteres potencialmente peligrosos antes de que estos datos se integren en la salida HTML, consultas SQL, logs, etc., lo que ayuda a evitar que los datos maliciosos se ejecuten en el navegador del usuario. Del mismo modo, para integrar datos en JavaScript, se recomienda usar *encoding* específico para JavaScript, y para incluir datos en URL, es crucial aplicar *encoding* de URL. Veamos algunos casos de códigos inseguros y cómo se podrían corregir:
  - **Incorporación directa de entrada del usuario en métodos HTTP:** evita la inserción directa de variables que contienen entrada de usuario ($_GET, $_POST, $_REQUEST, etc.) en el HTML. Por ejemplo:

```
<!-- Ejemplo en PHP, directamente en el HTML -->

<div><?php echo $_GET['comentario']; ?></div>
```

    Este código se podría mejorar con el uso de `htmlspecialchars`, que convierte caracteres especiales en entidades HTML, previniendo la ejecución de *script*s.

```
<div><?php echo htmlspecialchars($_GET['comentario'], ENT_QUOTES,
'UTF-8'); ?></div>
```

  - **Uso inseguro de InnerHTML:** evita asignaciones a innerHTML que utilizan variables no sanitizadas, lo que puede permitir la ejecución de código malicioso. Por ejemplo:

```
document.getElementById('div1').innerHTML = userInput;
```

Para corregir este código se puede usar `textContent`, que no interpreta el contenido como HTML, evitando la ejecución de cualquier *script* malicioso incluido en `userInput`.

```
document.getElementById('div1').textContent = userInput;
```

- **Manipulación de URL sin validación:** evita asignaciones de URL basadas en entradas no validadas o sanitizadas, lo que puede llevar a redirecciones maliciosas o ejecución de *script*s. Por ejemplo:

```
window.location = untrustedURL;
```

Validar las URL antes de su uso puede prevenir ataques de redirección y XSS.

```
if(validateURL(untrustedURL)) {
    window.location = untrustedURL;
}
```

- **Manipulación insegura del DOM:** evita el uso de `document.write` o funciones similares con entradas no controladas, lo que puede introducir código malicioso. Por ejemplo:

```
document.write(userInput);
```

En su lugar, crear un nodo de texto y al añadirlo al DOM evita la interpretación del userInput como código ejecutable.

```
let textNode = document.createTextNode(userInput);
document.body.appendChild(textNode);
```

El uso de herramientas como DOMPurify[9] u otras librerías para sanitizar el DOM también puede ser de gran ayuda.

```
let textNode = DOMPurify.sanitize(userInput);
```

- **Configurar *cookies* con los atributos `Secure` y `HttpOnly`:** conviene asegurarse de que las *cookies* sensibles se envíen solo a través de HTTPS utilizando el atributo `Secure` y estableciendo el atributo `HttpOnly` para evitar que las *cookies* sean accesibles a través de JavaScript, mitigando así el impacto potencial de un ataque XSS.
- **Uso de plantillas y *frameworks* seguros:** muchos *frameworks* modernos de desarrollo web, como React, Vue.js y Angular, ofrecen cierta protección automática contra XSS al escapar de manera predeterminada las entradas al incorporarlas en el DOM. Asegurarse de aprovechar estas características de seguridad es fundamental.

7.3. Pruebas continuas y auditorías de seguridad

[9] https://github.com/cure53/DOMPurify

Incluso después de implementar medidas de prevención, es vital realizar pruebas de seguridad de manera regular para detectar y remediar nuevas vulnerabilidades, como XSS, que puedan surgir debido a cambios en el código o en el entorno.

La integración de **pruebas de seguridad continuas** en el ciclo de vida del desarrollo de *software* (SDLC) mediante prácticas de DevSecOps es una estrategia habitual para garantizar que la seguridad sea una consideración primordial en todas las etapas del desarrollo. Incorporar pruebas de seguridad automatizadas en el proceso de integración y despliegue continuos (CI/CD), a través de herramientas especializadas en el SAST o DAST, puede ayudar a detectar rápidamente las vulnerabilidades XSS más obvias y contribuye a que los equipos puedan identificar y abordar las vulnerabilidades antes de que el código llegue a producción.

Además, la realización de **auditorías de seguridad**, tanto internamente como a través de terceros especializados, añade una capa adicional de confianza. Estas auditorías profundizan en la lógica de negocio de la aplicación y exploran vulnerabilidades que podrían no ser evidentes a través de herramientas automatizadas. Los expertos en seguridad que realizan estas auditorías aportan una perspectiva valiosa, identificando riesgos que requieren una comprensión contextual del funcionamiento de la aplicación y su entorno operativo. Estas auditorías pueden incluir pruebas de penetración manual, revisiones de configuración, evaluación de políticas de seguridad y prácticas de codificación, ofreciendo recomendaciones detalladas para la mitigación de riesgos identificados.

8. Conclusión

La prevención, detección y remediación de vulnerabilidades de *Cross-Site Scripting* (XSS) constituyen ejes centrales en el desarrollo seguro de las aplicaciones web. La protección efectiva contra XSS se basa en el uso estratégico de mecanismos como la política de mismo origen (SOP), compartición de recursos de orígenes cruzados (CORS) y políticas de seguridad de contenido (CSP). Estos enfoques técnicos, junto con la implementación de prácticas de codificación segura y la utilización de herramientas avanzadas de análisis y pruebas, conforman un escudo que puede reducir la brecha contra las amenazas de XSS.

La capacitación continua en prácticas de desarrollo seguro y la concienciación sobre los riesgos de seguridad representan componentes claves en la estrategia de defensa, pues dota a los equipos técnicos con el conocimiento y las herramientas necesarias para prevenir eficazmente ataques de XSS. En este sentido, **la preparación de un entorno de pruebas dedicado y aislado, como un laboratorio virtual, permite simular ataques y probar defensas en un contexto controlado y seguro, optimizando así el aprendizaje y la experimentación**.

Sin embargo, la eficacia de estas medidas de seguridad depende en gran medida de su implementación práctica y de la capacidad de las organizaciones para adaptarse a las peculiaridades tecnológicas específicas de sus plataformas. Desde entornos de desarrollo basados en diferentes lenguajes de programación hasta la diversidad de servidores web y configuraciones de aplicaciones, **la personalización y ajuste de las estrategias de seguridad a los contextos particulares son fundamentales para garantizar una protección efectiva contra XSS y otros vectores de ataque**.

En última instancia, **el enfoque para combatir XSS debe ser holístico y colaborativo**. Si bien en esta guía nos hemos centrado en las prácticas fundamentales que dependen en gran medida del lado del desarrollador, los administradores de sistemas y profesionales de TI también juegan un papel crucial en la **vigilancia y gestión de productos web de terceros** utilizados dentro de las organizaciones. La rápida aplicación de parches de seguridad y actualizaciones es esencial para cerrar las brechas que podrían ser explotadas por ataques XSS. Además, la **correcta configuración de los navegadores web** corporativos también contribuye significativamente a mitigar los riesgos asociados con el XSS. Por lo tanto, la seguridad frente a XSS trasciende las fronteras de la programación y el desarrollo web, convirtiéndose en una **responsabilidad compartida entre desarrolladores, administradores de sistemas, profesionales de TI y usuarios finales**. Cada grupo tiene un importante papel que desempeñar en la construcción y consumo de servicios webs más seguros.

La lucha contra XSS y la gestión de la seguridad web exigen un compromiso continuo con la mejora de la seguridad a través de la innovación tecnológica, el desarrollo de competencias especializadas y la adopción de una postura proactiva ante las ciberamenazas. **La seguridad de las aplicaciones web, frente a vulnerabilidades como XSS, no es solo un desafío técnico, sino un imperativo estratégico** para proteger la integridad, confidencialidad y disponibilidad de los activos digitales en el complejo ecosistema digital en el que vivimos.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia
incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

9. Acrónimos

- **AJAX:** Asynchronous JavaScript and XML
- **API:** Application Programming Interface
- **CI/CD:** Continuous Integration/Continuous Deployment
- **CORS:** Cross-Origin Resource Sharing
- **CSRF:** Cross-Site Request Forgery
- **CSP:** Content Security Policy
- **CSS:** Cascading Style Sheets
- **CWE:** Common Weakness Enumeration
- **DAST:** Dynamic Application Security Testing
- **DOM:** Document Object Model
- **DVWA:** Damn Vulnerable Web Application
- **HTML:** HyperText Markup Language
- **HTTP:** Hypertext Transfer Protocol
- **HTTPS:** Hypertext Transfer Protocol Secure
- **JSON:** JavaScript Object Notation
- **NAT:** Network Address Translation
- **PHP:** Hypertext Preprocessor
- **RASP:** Runtime Application Self-Protection
- **SAST:** Static Application Security Testing
- **SOP:** Same-Origin Policy
- **SQL:** Structured Query Language
- **SSRF:** Server-Side Request Forgery
- **URL:** Uniform Resource Locator
- **XSS:** Cross-Site Scripting

10. Bibliografía

Referencia - Título, autor, fecha y enlace web

[Ref.- 1] *Cross Site Scripting* (XSS), OWASP, https://owasp.org/www-community/attacks/xss/

[Ref.- 2] CAPEC-63: *Cross-Site Scripting* (XSS), MITRE, https://capec.mitre.org/data/definitions/63.html

[Ref.- 3] CWE-79: *Improper Neutralization of Input During Web Page Generation* ('*Cross-Site Scripting*'), MITRE, https://cwe.mitre.org/data/definitions/79.html

[Ref.- 4] *Cross-site Scripting* (XSS) cheat sheet, Portswigger, https://portswigger.net/web-security/cross-site-scripting/cheat-sheet

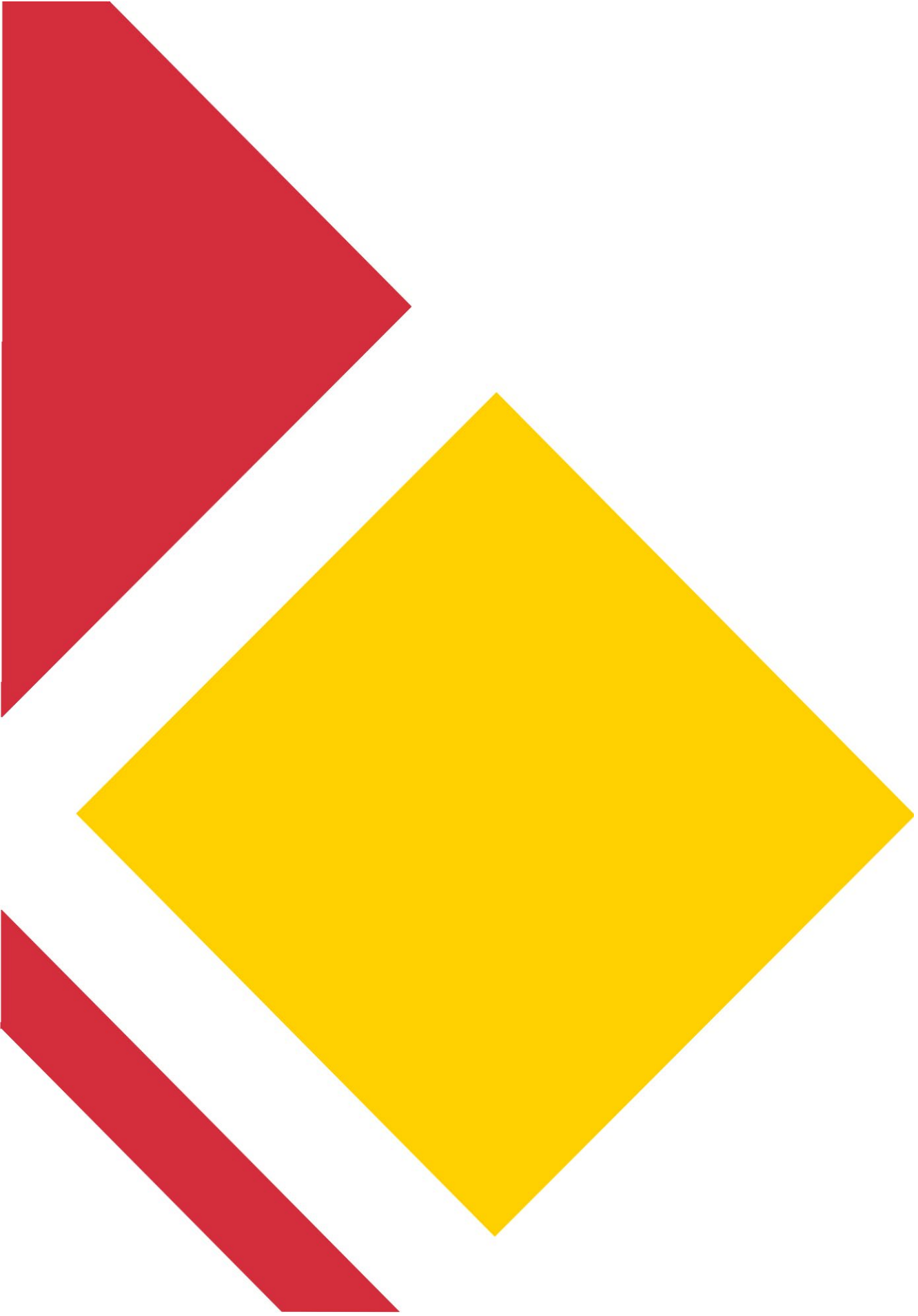
[Ref.- 5] *Current state of research on Cross-Site Scripting (XSS) – A systematic literature review,* I Hydara, ABM Sultan y otros, 2015, https://www.sciencedirect.com/science/article/pii/S0950584914001700

[Ref.- 6] *XSS Attacks: Cross Site Scripting Exploits and Defense*, J Grossman, 2007, https://books.google.es/books?hl=es&lr=&id=Imt5Crr0jJcC&oi=fnd&pg=PP2&dq=xss&ots=x84urYe0DX&sig=qT2ox6L9Pan6PMAM8i5ovdLxeZQ#v=onepage&q=xss&f=false

[Ref.- 7] *Cross-Site Scripting* (XSS) *attacks and defense mechanisms: classification and state-of-the-art*, S Gupta, BB Gupta, 2,

https://www.researchgate.net/publication/281823720_Cross-Site_Scripting_XSS_attacks_and_defense_mechanisms_classification_and_state-of-the-art017

[Ref.- 8] *Cross Site Scripting Prevention Cheat Sheet*, OWASP, https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#framework-security



Financiado por la Unión Europea
NextGenerationEU

GOBIERNO DE ESPAÑA
MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES

Plan de Recuperación, Transformación y Resiliencia

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD