Estudio sobre vulnerabilidades persistentes - **Path Traversal**

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

Julio 2026

INCIBE-CERT_ESTUDIO_VULNERABILIDADES_PERSISTENTES_*PATH*_TRAVERSAL_v1.0

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

Con el paso de los años, las aplicaciones web han crecido en complejidad, integrando una multitud de funcionalidades y servicios interconectados, lo que aumenta significativamente los vectores de ataque que deben ser considerados. Este hecho tecnológico introduce nuevos retos de seguridad a la hora de implementar una aplicación, donde una protección meticulosa en cada capa del sistema es esencial para mantener la integridad y seguridad de esta. **Entre los ataques más críticos y, a menudo, subestimados, se encuentra el *Path Traversal*, una vulnerabilidad que afecta directamente la interacción de las aplicaciones con el sistema de archivos subyacente.**

En un sistema de archivos, el control de acceso y la gestión de privilegios determinan quién puede ver, modificar o ejecutar archivos específicos, protegiendo así la integridad y confidencialidad de los datos almacenados. Cada archivo y directorio puede tener permisos específicos, como lectura, escritura y ejecución, que permiten limitar las acciones que los usuarios pueden realizar. Estos controles son esenciales para evitar accesos no autorizados y la fuga de información sensible, ya que garantizan que solo los usuarios autorizados puedan interactuar con archivos críticos. Así, una adecuada configuración de permisos en el sistema de archivos ayuda a mitigar riesgos de seguridad y a proteger los datos frente a amenazas internas y externas.

La vulnerabilidad de *Path Traversal* (o recorrido de directorios) **ocurre cuando un sistema permite que un atacante manipule las rutas de acceso a archivos del sistema mediante entradas de usuario**. En lugar de limitarse a las rutas previstas, el atacante puede usar secuencias especiales para acceder a otros directorios en la jerarquía de archivos. Esto le da la posibilidad de explorar o manipular archivos y directorios que deberían estar restringidos. A pesar de ser una vulnerabilidad conocida, **su explotación persiste debido a la falta de validación adecuada en muchas aplicaciones**, lo que expone a los sistemas a riesgos significativos, como la exfiltración de datos, la elevación de privilegios o la ejecución de *malware*, especialmente en aplicaciones o componentes heredados que no hayan tenido un mantenimiento adecuado.

Los desarrolladores de aplicaciones y administradores de sistemas deben implementar medidas de seguridad eficaces contra esta vulnerabilidad para proteger la integridad de los sistemas. Esto requiere un enfoque robusto de validación de entradas y restricciones en los permisos de acceso a archivos, aplicando siempre las mejores prácticas en la gestión de permisos. Si no se aborda correctamente, un ataque puede resultar en fugas de información confidencial, modificaciones no autorizadas en el sistema y, en algunos casos, compromisos a nivel de servidor.

Esta guía, dirigida tanto a desarrolladores como a profesionales de la seguridad, subraya la importancia de entender y mitigar los riesgos asociados con el *Path Traversal*.

.

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

2. Organización del documento

El informe comienza con la sección **3.- Introducción**, donde se exploran los conceptos fundamentales del ataque de *Path Traversal*, explicando cómo este tipo de vulnerabilidad puede ser explotado para acceder a archivos o directorios no autorizados en un servidor. Se resalta la creciente necesidad de proteger las aplicaciones web frente a estas amenazas en un panorama digital cada vez más complejo y dinámico.

En la sección **4.- Tipologías de análisis y herramientas**, se ofrece una visión detallada de las estrategias y tecnologías que permiten detectar y evaluar posibles fallos de seguridad relacionados con *Path Traversal*. Tanto los enfoques de análisis estático como dinámico son abordados, y se enumeran las principales herramientas que ayudan a identificar estos riesgos de manera eficiente.

En la sección **5.- Ejemplos de análisis de vulnerabilidades**, se expone un ejemplo práctico que demuestra cómo se pueden descubrir y analizar vulnerabilidades en condiciones reales dentro de este entorno seguro.

En la sección **6.- Prevención y buenas prácticas**, se detallan las medidas preventivas que pueden implementarse para reducir el riesgo de ataques de *Path Traversal*. Se subraya la importancia de seguir principios de codificación segura, definir políticas de control de acceso rigurosas y mantener un proceso continuo de revisión y pruebas de seguridad para garantizar la protección de los datos.

Finalmente, la sección **7.- Conclusión**, reúne los principales hallazgos del estudio y destaca la importancia de adoptar una postura de seguridad activa, integrando buenas prácticas y auditorías regulares durante todo el ciclo de vida del desarrollo de aplicaciones.

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 protección de las aplicaciones web es clave para prevenir ataques y violaciones de seguridad, y para mantener la integridad y confianza de los usuarios. **Entre las vulnerabilidades que merecen especial atención actualmente en el panorama de la ciberseguridad se encuentra *Path Traversal*.**

El estándar canónico para clasificar la vulnerabilidad de salto de directorio es la entrada CWE-22 de MITRE **(CWE-22[1]),** la cual detalla los riesgos de una limitación inadecuada de rutas, incluyendo variantes relacionadas como CWE-23, CWE-36 y CWE-73 y abarcando métodos de detección a través de análisis estático y dinámico, proponiendo la validación estricta, la canonicalización de rutas y la implementación de listas blancas como mitigaciones fundamentales.

Este tipo de vulnerabilidad ha sido clasificado consistentemente como una amenaza recurrente y ha protagonizado incidentes importantes en diversas organizaciones, por eso aparece en la posición 8 del Top 25 de vulnerabilidades persistentes[2] de Common Weakness Enumeration (CWE), y dentro de la lista de Open Worldwide Application Security Project (OWASP) TOP 10 2021, en su categoría A01:2021 – Pérdida de Control de Acceso[3].

La vulnerabilidad de *Path Traversal* se produce cuando una aplicación permite a los usuarios manipular rutas de archivos, lo que les brinda la posibilidad de salir del directorio autorizado y acceder a recursos sensibles del sistema, como contraseñas, *logs* del sistema, configuraciones, o cualquier clase de información personal sensible que almacenen los sistemas. Este tipo de ataques suele ocurrir cuando no se validan correctamente las entradas proporcionadas por los usuarios, permitiendo el uso de secuencias para navegar por el sistema de archivos. El impacto puede ser devastador, desde la exposición de información confidencial hasta el compromiso completo del sistema, lo que puede llevar a la ejecución de código malicioso o la destrucción de datos. Organizaciones de todo tipo pueden verse afectadas si no implementan adecuadamente medidas de control como la validación estricta de entradas, o la restricción de accesos a determinados directorios.

Dado el alcance y la simplicidad con la que puede explotarse *Path Traversal*, es importante comprender a fondo cómo identificar y mitigar esta vulnerabilidad. Exploraremos cómo funciona este tipo de ataque, cómo los atacantes lo aprovechan y, lo más importante, qué medidas de defensa pueden adoptar las organizaciones para protegerse eficazmente.

3.1. Definición de *Path Traversal*

En una aplicación, **un *path* o ruta es una cadena de texto que indica la ubicación específica de un archivo o directorio dentro del sistema de archivos**. Este *path* permite a la aplicación acceder, modificar o ejecutar archivos según sea necesario para su

[1] https://cwe.mitre.org/data/definitions/22.html

[2] https://cwe.mitre.org/top25/archive/2023/2023_stubborn_weaknesses.html

[3] https://owasp.org/Top10/es/A01_2021-Broken_Access_Control/

funcionamiento. Los *paths* son fundamentales para cualquier operación que implique leer o escribir datos en el sistema de archivos, y pueden ser absolutos o relativos.

- Un ***path* absoluto** proporciona la ruta completa desde la raíz del sistema de archivos hasta el archivo o directorio deseado, como `/home/usuario/documentos/archivo.txt`. Esta ruta es fija y válida desde cualquier punto del sistema.
- Un ***path* relativo**, en cambio, se define en relación con el directorio actual en el que se encuentra la aplicación. Por ejemplo, `documentos/archivo.txt` se refiere a un archivo en el subdirectorio documentos del directorio actual de trabajo. Para moverse entre directorios, los *paths* relativos pueden incluir secuencias especiales como `../`, que indica subir un nivel en la jerarquía de directorios.

La sintaxis del *path* en una aplicación depende del sistema operativo en el que esté ejecutándose, ya que cada sistema utiliza convenciones diferentes para estructurar las rutas de acceso a archivos y directorios. En sistemas tipo Unix/Linux y macOS, las rutas usan el carácter '/' como separador de directorios y empiezan desde la raíz del sistema (`/`). En Windows, en cambio, las rutas utilizan el carácter '`\`' como separador de directorios, y los *paths* absolutos suelen empezar con una letra de unidad, como `C:\Usuarios\Usuario\Documentos\archivo.txt`. Windows también permite usar `/` en algunas situaciones, pero la barra invertida `\` es el estándar.

La vulnerabilidad ***Path Traversal*** se refiere a la posibilidad que los usuarios manipulen las rutas de archivos de forma no controlada, accediendo a archivos y directorios fuera del entorno restringido o previsto por el sistema. **El ataque se basa en el uso de secuencias especiales dentro de la ruta de archivo. Esto puede realizarse de dos formas: a través de rutas relativas o rutas absolutas, para acceder a áreas sensibles del sistema**.

- ***Path Traversal* Relativo** (CWE-23)[4]: aquí, el atacante usa secuencias como `../` para moverse en relación con el directorio en el que está la aplicación, accediendo a archivos a partir de esa ubicación inicial. Por ejemplo, si la aplicación está ubicada en `/home/usuario/app/`, un atacante podría usar una entrada relativa como `../../../etc/passwd` para salir de los directorios controlados y alcanzar el archivo `passwd` en `/etc`.

Ilustración 1: Ejemplo de petición legítima al servidor

[4] https://cwe.mitre.org/data/definitions/23.html

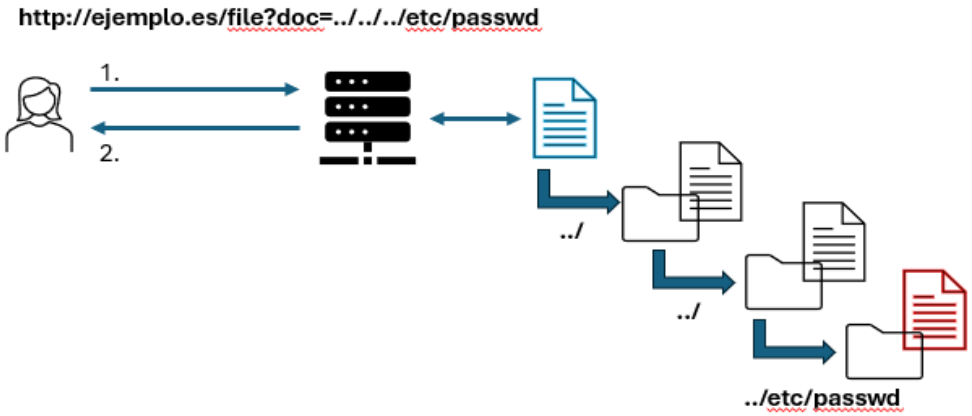

Ilustración 2: Ejemplo de petición maliciosa Path Traversal Relativo

- ***Path Traversal* Absoluto** (CWE-36)[5]: en este caso, el atacante emplea rutas completas que comienzan desde la raíz (`/`) del sistema de archivos, intentando acceder directamente a ubicaciones específicas sin importar el directorio en el que esté la aplicación. Permite a los atacantes especificar una **ruta absoluta**, es decir, una ruta completa desde el directorio raíz del sistema de archivos, como `/etc/passwd` o `C:\Windows\system32\`.

En la mayoría de los casos, *Path Traversal* ocurre cuando las aplicaciones web permiten a los usuarios introducir rutas de archivos a través de parámetros URL o formularios sin la validación adecuada. En el siguiente fragmento de código podemos encontrar una funcionalidad que permite el acceso a archivos especificando el nombre concreto de un archivo en la propia URL.

```
$document = $_GET['doc'];
$filepath = '/var/www/html/documentos/'.$documento;

if (file_exists($filepath)) {
    readfile($filepath);
} else {
    echo 'Archivo no encontrado';
}
```

En este caso, el valor de la variable `$document` proviene de una solicitud HTTP GET, lo que significa que el usuario puede controlar su contenido directamente mediante la URL. El código de la aplicación genera una ruta concatenando el valor de `$document` con un directorio base (`/var/www/html/documentos/`), creando así una ruta completa hacia el archivo al que intenta acceder la aplicación. Por ejemplo, si el usuario especifica `archivo.txt` en la URL, el valor resultante de `$filepath` sería `/var/www/html/documentos/archivo.txt`. Luego, el código verifica si el archivo existe mediante `file_exists($filepath)`, y si es así, la función `readfile($filepath)` lee y muestra el contenido del archivo.

[5] https://cwe.mitre.org/data/definitions/36.html

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

Como podemos intuir, el código presenta un *Path Traversal* Relativo. Un atacante podría aprovecharlo para modificar la URL y manipular la ruta de acceso. Si usa `../../../../etc/passwd` como valor de `$document`, el sistema construirá una ruta como `/var/www/html/documentos/../../../../etc/passwd`. Al resolver esta ruta, el sistema se dirigirá al archivo `/etc/passwd`, fuera del directorio `documentos`. Como el archivo existe en cualquier configuración por defecto Linux y asumiendo que el servidor tiene permisos de lectura, `readfile()` lo mostrará, exponiendo contenido sensible de las contraseñas sistema.

3.2. Postexplotación del *Path Traversal*

Una vez identificada una vulnerabilidad de *Path Traversal*, las actividades de post-explotación que pueden llevarse a cabo son variadas y dependen un poco de otras vulnerabilidades encadenadas. A continuación, se detallan las acciones que los atacantes podrían realizar tras haber comprometido un sistema a través de *Path Traversal*:

- **Acceso no autorizado a archivos sensibles:** una de las primeras acciones que realiza un atacante es aprovechar el *Path Traversal* para acceder a archivos fuera del directorio permitido por la aplicación. Archivos críticos como configuración del servidor (`config.php`, `web.config`), credenciales de bases de datos y claves API almacenadas en archivos de entorno `(.env)`, o incluso archivos del sistema como `/etc/passwd` en Unix/Linux, que contienen información sobre usuarios y contraseñas. También puede comprometer información de identificación personal (PII), como historiales de transacciones, direcciones y otros datos sensibles de usuarios, violando normativas de privacidad como GDPR.
- **Ejecución de código remoto (RCE):** puede incluir técnicas como *Local File Inclusion* (LFI) y *Remote File Inclusion* (RFI), que permiten al atacante cargar y ejecutar archivos maliciosos para obtener control del sistema. Alternativamente a la carga de ficheros maliciosos, en otros casos, si el atacante logra acceder a archivos que contienen *scripts* ejecutables, podría modificarlos para incluir en ellos una carga maliciosa. Un ejemplo claro de esto es el ataque conocido como *Log Poisoning* en el que el atacante introduce comandos dentro de un *log* para que luego a través del acceso a este *log* se ejecute.
- **Escalado de privilegios**: el atacante puede acceder a archivos o directorios que poseen permisos elevados y aprovechar esta ventaja para consolidar su control sobre el sistema. Por ejemplo, el atacante puede extraer credenciales de usuarios con privilegios superiores o modificar los parámetros de seguridad, facilitando su propio acceso y su capacidad para realizar operaciones con permisos elevados. Además, si el atacante encuentra *scripts* de inicio o archivos ejecutables con permisos elevados, puede insertar código malicioso o modificar esos archivos para ejecutarse automáticamente con privilegios administrativos en cada inicio del sistema, manteniendo así un acceso persistente y autorizado en futuras sesiones.

Este tipo de actividades de post-explotación pueden tener un impacto devastador en la infraestructura de una organización, afectando tanto a la integridad del sistema como a la confidencialidad y disponibilidad de los datos. La implementación de controles de seguridad sólidos, como la validación estricta de entradas y la revisión continua de permisos de acceso, es fundamental para mitigar los riesgos asociados a *Path Traversal* y garantizar la resiliencia del sistema frente a este tipo de ataques.

3.3. Nuevos escenarios de exposición

Aunque *Path Traversal* es una vulnerabilidad conocida desde hace años, sigue apareciendo con frecuencia en aplicaciones y arquitecturas actuales. La evolución de las tecnologías no ha eliminado este tipo de debilidad; simplemente ha trasladado su impacto a nuevos escenarios donde el acceso al sistema de archivos continúa siendo un elemento crítico.

Hoy en día es habitual encontrar aplicaciones basadas en microservicios, contenedores, plataformas cloud, APIs REST o soluciones que incorporan capacidades de Inteligencia Artificial. En todos estos entornos se gestionan rutas hacia modelos, conjuntos de datos, archivos temporales, repositorios de artefactos o recursos compartidos. Cuando dichas rutas pueden verse influenciadas, directa o indirectamente, por datos externos y no se validan correctamente, vuelven a aparecer los mismos problemas descritos por CWE-22[8].

Por este motivo, independientemente de la tecnología utilizada, resulta recomendable mantener una serie de medidas básicas de protección:

- Resolver y canonicalizar las rutas antes de cualquier validación o acceso al sistema de archivos.
- Limitar el acceso únicamente a directorios previamente autorizados (*allowlists*).
- Aplicar el principio de mínimo privilegio a los procesos y servicios.
- Restringir el acceso al sistema de archivos únicamente a los recursos estrictamente necesarios.
- Aislar adecuadamente los distintos componentes de la aplicación cuando se utilicen contenedores o arquitecturas distribuidas.

En definitiva, la aparición de nuevas tecnologías no modifica la naturaleza de esta vulnerabilidad, pero sí amplía los escenarios donde puede manifestarse. Por ello, resulta especialmente importante mantener los mismos principios de desarrollo seguro independientemente de la plataforma utilizada.

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

4. Tipologías de análisis y herramientas

Para detectar y confirmar la existencia de vulnerabilidades de *Path Traversal*, los evaluadores de seguridad pueden utilizar varias técnicas mediante herramientas de análisis de seguridad, como SAST (*Static Application Security Testing*), DAST (*Dynamic Application Security Testing*) y pruebas de penetración (*pentesting*).

4.1. Técnicas de test

Estas técnicas buscan identificar patrones de manipulación de rutas que permiten acceder a archivos no autorizados o sensibles y evadir controles de seguridad básicos:

- El SAST o **análisis estático**, se enfoca en revisar el código fuente de la aplicación sin ejecutarla, con el objetivo de identificar patrones de programación que podrían permitir la explotación de *Path Traversal*. Se evalúa si el código aplica filtros que bloqueen secuencias como `../`, `\`, o caracteres especiales, y si las entradas de usuario se normalizan adecuadamente para evitar la manipulación de rutas hacia directorios no autorizados. Además, se analiza si el código evita la concatenación directa de rutas y en su lugar utiliza funciones seguras para la construcción de rutas, como `path.join` en Node.js o clases específicas de manejo de archivos en otros lenguajes. Finalmente, se observa el uso de funciones de acceso a archivos (`open()`, `readfile()`) para confirmar que no reciben rutas basadas en entradas sin una validación exhaustiva, ya que estos puntos de acceso son vulnerables a *Path* Traversal si no se controlan correctamente.

- El DAST o **análisis dinámico** implica ejecutar la aplicación y observar cómo maneja las rutas de archivos en tiempo real. Durante este proceso, se busca cómo la aplicación responde a entradas manipuladas en tiempo de ejecución, probando inyecciones de secuencias como `../` en parámetros de URLs, formularios y encabezados para intentar navegar fuera del directorio permitido. También se prueban codificaciones en URL y Unicode para evadir filtros de seguridad que bloquean estas secuencias directamente, verificando si el servidor las decodifica y permite el acceso a archivos críticos. Los evaluadores intentan acceder a archivos conocidos y sensibles, como `/etc/passwd` en Unix/Linux, para confirmar la existencia de la vulnerabilidad si la aplicación responde con información inesperada. Además, en aplicaciones con carga de archivos, el DAST permite verificar si el *Path Traversal* puede combinarse con archivos cargados maliciosamente para ejecutar código remoto (RCE), mientras que el análisis de mensajes de error revela información adicional sobre la estructura interna y posibles rutas de acceso no autorizadas.

- El *pentesting* o **pruebas de penetración** son esenciales para comprender cómo las vulnerabilidades de *Path Traversal* podrían ser explotadas en un entorno real. Estas pruebas, a menudo realizadas manualmente o con herramientas especializadas, implican la simulación de ataques dirigidos que intentan aprovechar fallos de validación de rutas. Los *pentesters* también deben evaluar cómo responde 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

a las manipulaciones de rutas y si se aplican correctamente las restricciones de acceso a nivel de archivo.

- Una de las variantes más conocidas de *Path Traversal* es ***Zip Slip***[10], un patrón de vulnerabilidad que puede aparecer durante la extracción de archivos comprimidos (ZIP, TAR, JAR u otros formatos similares) cuando la aplicación no valida correctamente las rutas contenidas en cada una de las entradas del archivo.
  Si un fichero comprimido contiene rutas manipuladas, por ejemplo mediante secuencias como `(../)`, la aplicación podría escribir archivos fuera del directorio previsto si concatena dichas rutas sin realizar previamente un proceso adecuado de normalización y canonicalización. Dependiendo del contexto, este comportamiento puede permitir la sobreescritura de archivos de configuración, bibliotecas, ejecutables u otros recursos críticos del sistema.
  Aunque el mecanismo de explotación es diferente al de un *Path Traversal* tradicional sobre una aplicación web, el problema subyacente es el mismo: la falta de una validación adecuada de las rutas antes de acceder al sistema de archivos.
  La principal medida de mitigación consiste en resolver y canonicalizar la ruta completa antes de extraer cualquier archivo, comprobando posteriormente que la ubicación resultante permanece dentro del directorio autorizado. Esta validación debería complementarse con controles adicionales sobre los nombres de archivo, listas de directorios permitidos y la aplicación del principio de mínimo privilegio durante todo el proceso de extracción.
  El patrón *Zip Slip*, ampliamente documentado por la comunidad de seguridad, ha afectado históricamente a múltiples bibliotecas y herramientas de diferentes lenguajes de programación, poniendo de manifiesto que este tipo de errores continúa siendo una fuente habitual de vulnerabilidades cuando no se aplican correctamente las medidas de validación de rutas.

4.2. Batería de test

A continuación, planteamos algunos casos de test que se pueden definir como batería independientemente de la técnica utilizada. Aunque el planteamiento general es aplicarlo a *Path Traversal* relativo, se puede aplicar igualmente a la versión de *path* absoluto.

Manipulación de rutas con caracteres reservados para escapar de los directorios

Ocurre cuando se utiliza secuencias de retroceso de directorios, como `../` en sistemas basados en Unix o `..\` en Windows, para acceder a archivos fuera del directorio previsto. Las aplicaciones vulnerables no validan adecuadamente la entrada del usuario, permitiendo que los evaluadores manipulen la ruta del archivo. Este tipo de ataque puede llevar al acceso no autorizado a archivos sensibles como configuraciones, credenciales o incluso archivos binarios del sistema. El funcionamiento técnico se puede definir en los aspectos siguientes:

- Los sistemas operativos utilizan secuencias como `../` para moverse 'hacia arriba' en la jerarquía de directorios.
- Un evaluador inserta estas secuencias en los parámetros que la aplicación utiliza para construir rutas de archivos.

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

- Si la aplicación no valida correctamente o sanitiza estas entradas, el evaluador puede navegar hacia directorios superiores y acceder a archivos fuera del directorio autorizado.

```
GET /download?file=../../../../etc/passwd
```

En este ejemplo, el evaluador intenta acceder al archivo `/etc/passwd`, que contiene información crítica sobre los usuarios del sistema.

Uso de codificaciones URL o Unicode para evadir controles

En muchos casos, las aplicaciones implementan filtros o mecanismos de validación para evitar el uso directo de secuencias como `../`. Sin embargo, los evaluadores pueden valorar su seguridad utilizando **codificaciones de URL** o **Unicode**. Estas técnicas permiten representar los mismos caracteres de forma codificada, dificultando la detección por los filtros. Los navegadores y servidores web suelen decodificar estas secuencias antes de procesar la solicitud, lo que da lugar a la ejecución del ataque a pesar del filtrado. En cuanto al funcionamiento técnico, se detallan los siguientes aspectos:

- La codificación de URL transforma caracteres reservados como `'.'` y `'/'` en su representación hexadecimal (`%2e, %2f`).
- Unicode permite usar secuencias codificadas que representan los mismos caracteres.
- Si la aplicación no decodifica y valida correctamente la entrada, el servidor decodificará la secuencia después del filtrado, permitiendo que el ataque funcione.

```
GET /download?file=%2e%2e%2f%2e%2e%2f%2e%2e%2fetc%2fpasswd
```

Aquí, la secuencia `'../../'` está codificada como `'%2e%2e%2f'` para evadir los filtros de seguridad.

Explotación de la vulnerabilidad junto con cargas de archivos

En algunos casos, el ***Path Traversal*** puede combinarse con otras vulnerabilidades, como la **carga insegura de archivos**. Un evaluador puede aprovechar un fallo en la validación de la ruta y manipular archivos que ya se han subido al servidor, o cargar un archivo malicioso en una ubicación accesible a través de una ruta manipulada. Este enfoque puede llevar a la ejecución remota de código, sobreescritura de archivos o incluso la ejecución de *scripts* maliciosos en el servidor. En este caso, el funcionamiento técnico se detalla en:

- Un evaluador carga un archivo malicioso (como una *shell* web o un archivo ejecutable) en el servidor.
- Utilizando *Path Traversal*, manipula las rutas para acceder y ejecutar el archivo desde un directorio fuera del acceso autorizado.
- Las aplicaciones que no validan correctamente las rutas de acceso permiten esta combinación de ataques.

Ejemplo**:** El evaluador carga un archivo malicioso llamado `shell.php` en un directorio permitido. Luego utiliza *Path Traversal* para acceder y ejecutar el archivo:

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

```
GET /files/view?file=../../uploads/shell.php
```

Uso de caracteres nulos u otros trucos para engañar la validación de rutas mediante truncamiento

El **truncamiento de rutas** es una técnica en la que los evaluadores emplean caracteres especiales, como el carácter nulo (`%00`), para manipular las rutas de archivos. En algunos lenguajes de programación, como C, las funciones de manejo de cadenas tratan el carácter nulo como un terminador de cadena, lo que provoca que se ignore todo lo que venga después. Los evaluadores pueden usar esta técnica para engañar a las aplicaciones que validan las rutas basándose en extensiones de archivos o patrones específicos, permitiendo que se ejecuten archivos no deseados. Su funcionamiento consta de los siguientes aspectos:

- El evaluador inserta un carácter nulo en la ruta para truncar la cadena de archivo en el lado del servidor.
- La aplicación cree que el archivo tiene una extensión o formato permitido, pero el carácter nulo hace que el servidor procese una ruta diferente.
- Esta técnica es especialmente efectiva en aplicaciones mal diseñadas que utilizan funciones vulnerables al truncamiento.

Ejemplo**:** El evaluador intenta acceder al archivo `/etc/passwd`, pero la aplicación solo permite descargar archivos con la extensión `.jpg`. Al usar un carácter nulo `%00`, se engaña o trunca la cadena después de `/etc/passwd`, con lo que la aplicación ignora la extensión permitiendo el acceso al archivo:

```
GET /download?file=../../etc/passwd%00.jpg
```

Las técnicas descritas deben utilizarse únicamente en entornos autorizados y con fines de auditoría o investigación/mitigación.

5. Ejemplos de análisis de vulnerabilidades
Path Traversal

Como ya se ha comentado en las secciones anteriores de este documento, se van a usar los laboratorios que aporta *PortSwigger* para aprender y practicar con los diferentes ámbitos de la ciberseguridad. También se va a utilizar la herramienta de esta misma compañía, *Burpsuite*, en su versión gratuita, para interceptar y alterar las peticiones que serán enviadas al servidor.

Para el estudio práctico de esta vulnerabilidad, se pondrá el foco sobre los dos casos que más se dan en aplicaciones de manera aislada.

Manipulación de rutas con ../ para escapar de los directorios

Para comenzar con este ejercicio, se debe acceder al primero de los laboratorios de la temática *Path Traversal*. Al acceder al laboratorio, se aprecia el botón para acceder al ejercicio práctico del laboratorio.

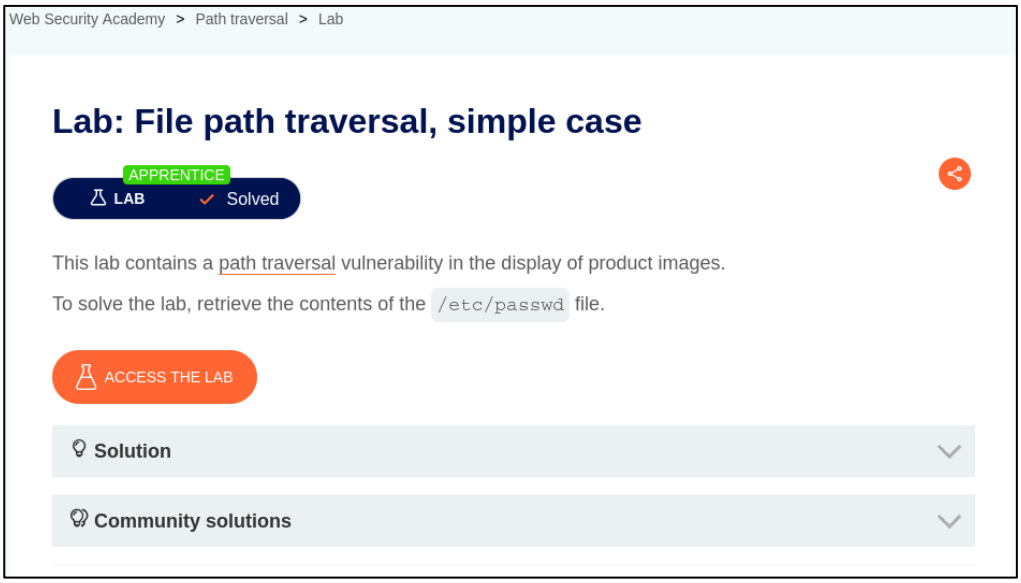

Ilustración 3: Acceso al primero de los laboratorios

Al acceder al laboratorio, tras pulsar el correspondiente botón, se puede observar como se muestra una aplicación web que corresponde a una tienda online. En la vista principal de la aplicación se observa un catálogo de diferentes productos.

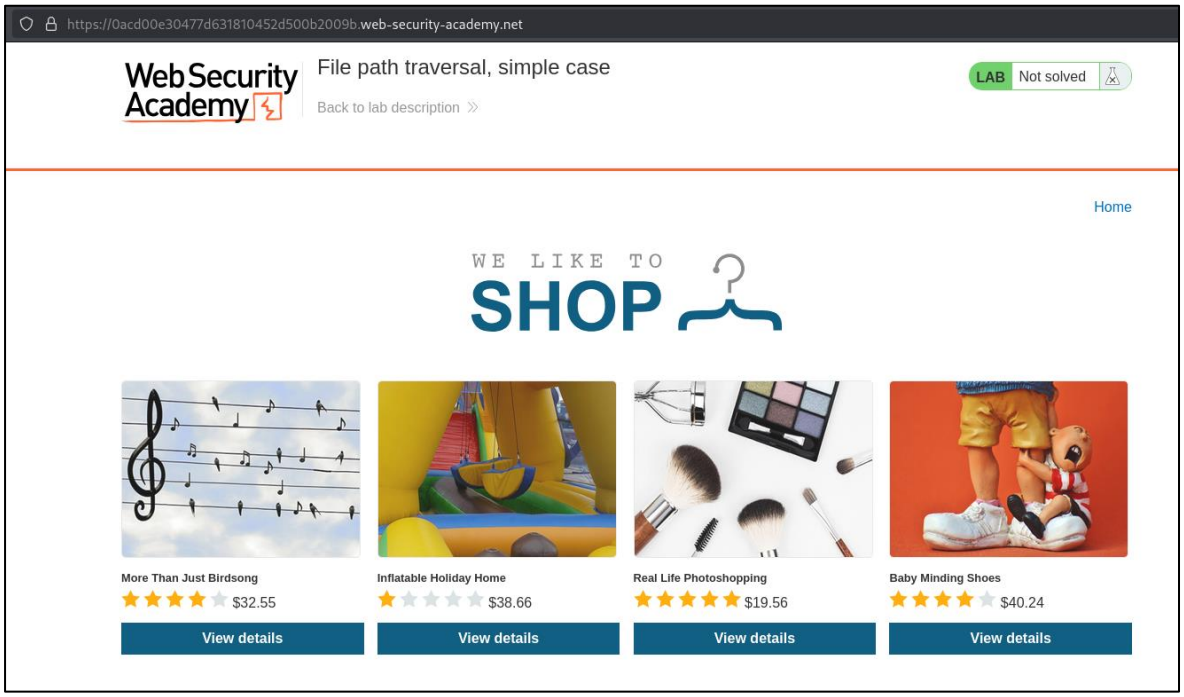

Ilustración 4: Página principal aplicación vulnerable

El primer paso que se debe dar siempre que se proceda a evaluar la seguridad de una aplicación web, es navegar por dicha aplicación poniendo atención en las rutas por las que vamos navegando. En este caso al estar usando la aplicación Burpsuite como *proxy* web, aún sin estar activa la intercepción de peticiones, el proceso de registro de rutas por las que se navega, se realiza automáticamente. El siguiente paso importante que se debe de dar al evaluar una aplicación web, es inspeccionar el código fuente accesible de la web para identificar rutas desde las que se carguen archivos y posibles fallos de programación o comentarios que filtren información.

Al navegar por la web, se aprecia que esta sólo tiene dos vistas, una en la que se muestra el catálogo de productos y la vista de detalle de cada uno de los productos, en esta última se puede observar que la URL de navegación contiene un parámetro que recibe un valor para identificar el objeto y que en base a ese valor muestra la descripción de cada uno de los objetos del catálogo.

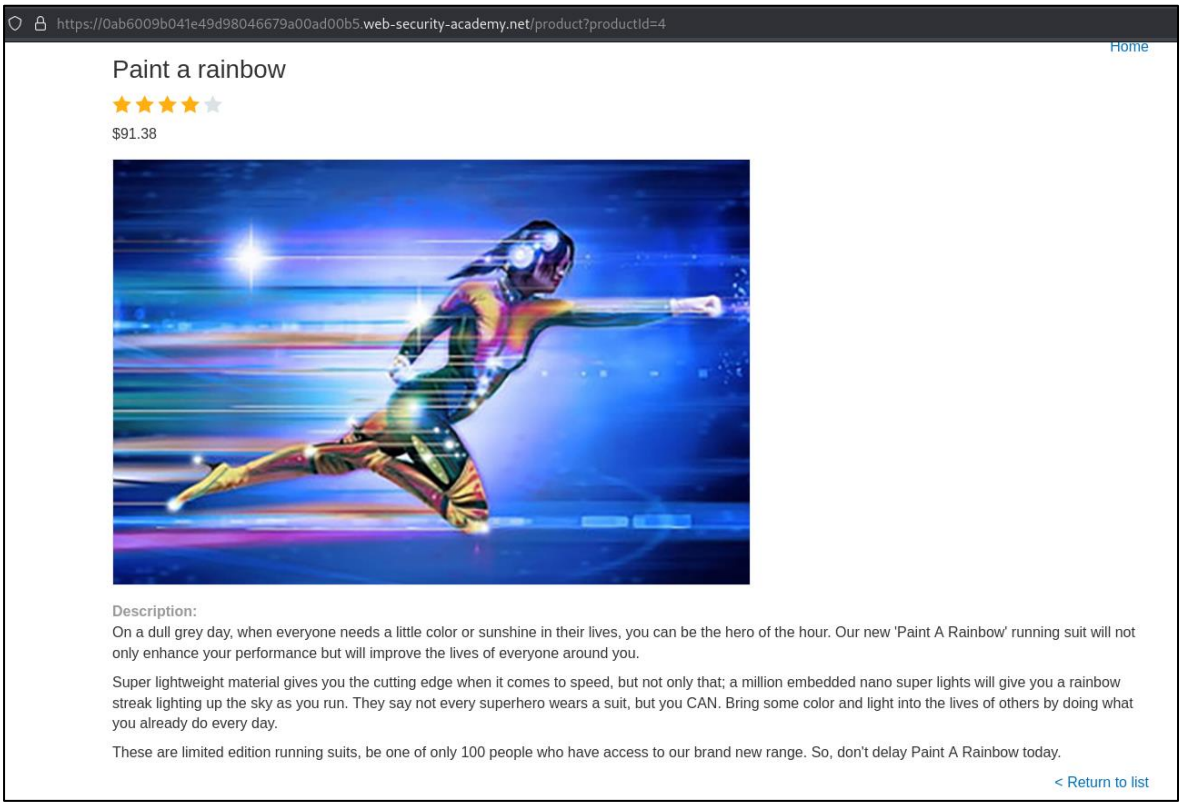

Ilustración 5: Descripción detallada de producto

Al alterar el valor del `id` del producto se aprecia que el servidor devuelve el mensaje de que el id del producto es inválido, por lo que es un indicativo de que existe algún control del formato del valor de la variable `productId`.

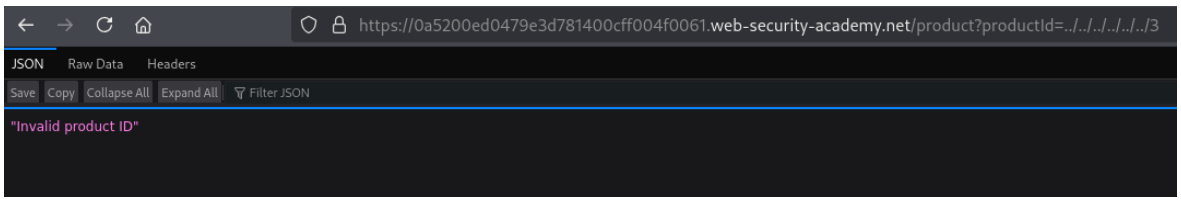

Ilustración 6: Intento de inyección en productId

Inspeccionando el código fuente de la web, se observa que las imágenes se cargan desde una ruta que sigue el mismo formato que la carga de productos, por lo que se puede deducir que también es una llamada a la API.

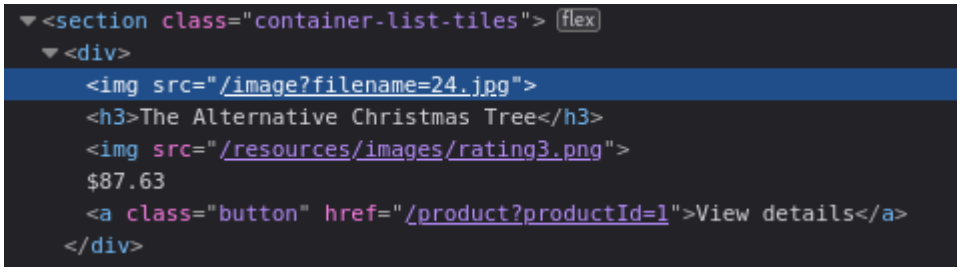

Ilustración 7:Código fuente carga de imágenes

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

En este punto vamos a observar las peticiones a través del Burpsuite. Para esto, se debe activar la opción de interceptar de Burpsuite y en el navegador introducir la URL de la imagen. Al hacer esto se ve como la petición llega al *proxy* web, si se quiere capturar también la respuesta se debe hacer clic derecho sobre la petición e ir a la opción `Do intercept`.

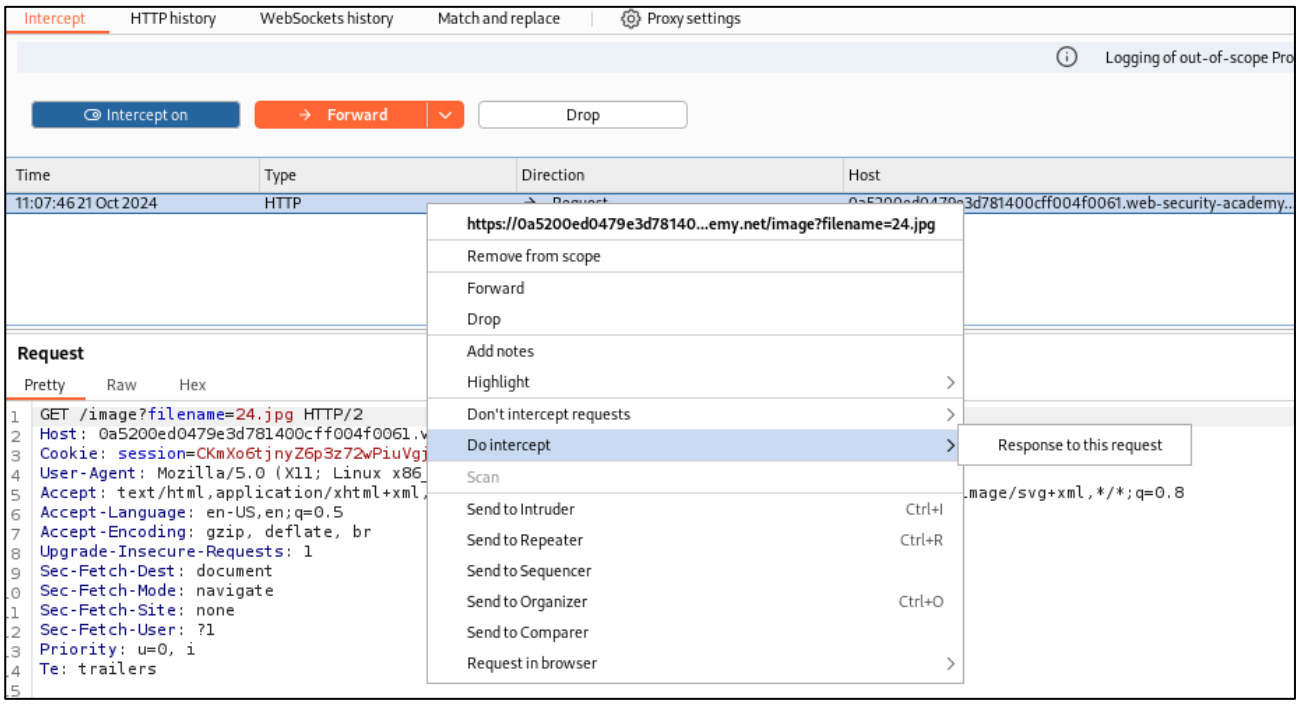

Ilustración 8: Captura de respuesta a una petición

Una vez pulsada la opción `Response to this request` se debe pulsar sobre `Forward` para enviar la petición al servidor y acto seguido el *proxy* capturará la respuesta del servidor en la que se puede ver la imagen en crudo como si fuera una cadena de texto.

Ilustración 9: Captura de respuesta con una imagen de producto

Se puede ver como la aplicación no permite mediante funcionalidades llegar a las rutas de imágenes, por lo que es probable que no estén sujetas a los mismos controles de seguridad que las de los productos. Para comprobarlo se puede inyectar el *payload* como valor de la variable `filename`. Al hacer esto se aprecia que no existe ningún tipo de control en la entrada de usuario para esta variable y que el servidor devuelve directamente el archivo solicitado.

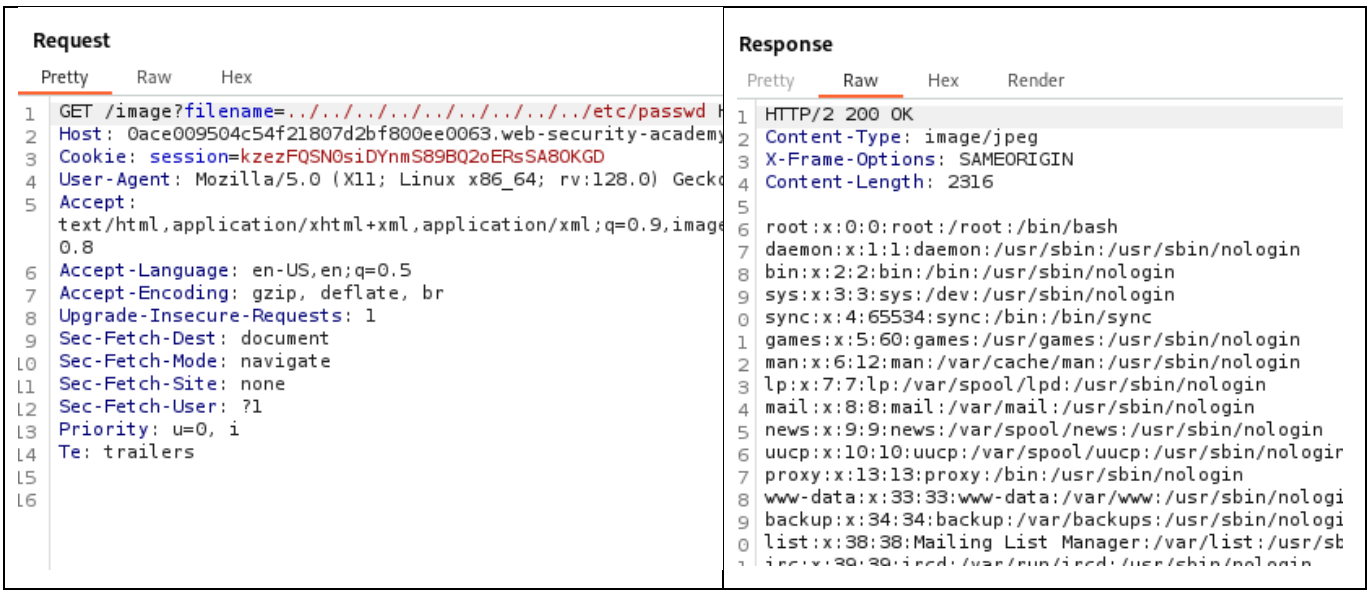

Ilustración 10: Explotación del Path Traversal

Uso de caracteres nulos u otros trucos para engañar la validación de rutas

En este caso el ejercicio práctico tendrá varias medidas de seguridad implementadas y se detallará como el uso de caracteres nulos pueden ayudar a un evaluador a saltarse medidas de protección pobres o mal implementadas. Para comenzar el ejercicio se tiene que acceder al laboratorio correspondiente.

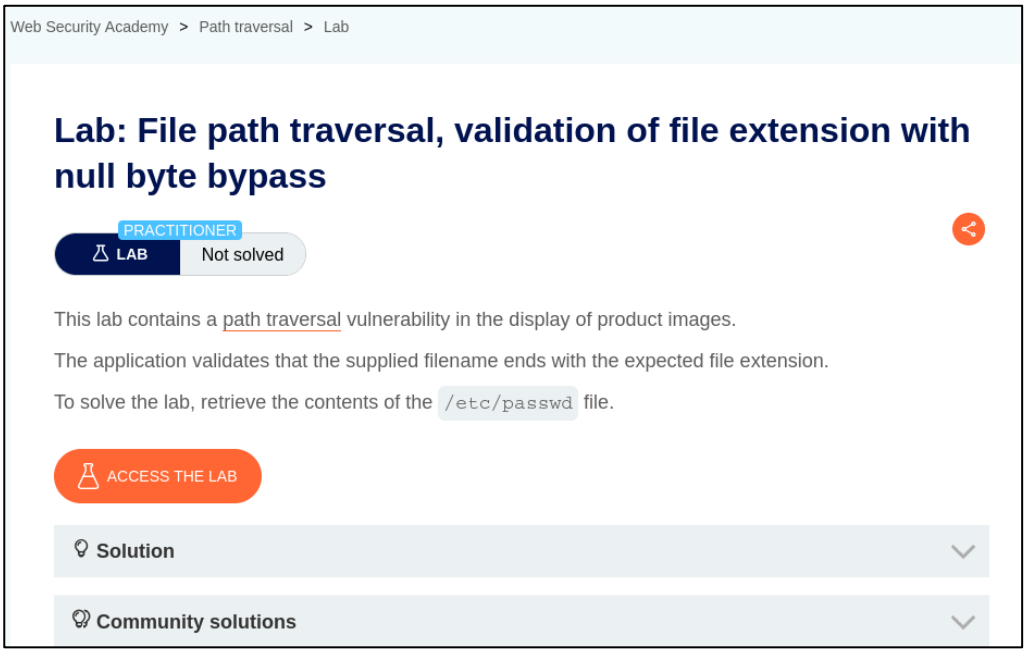

Ilustración 11: Laboratorio con medidas de protección

Al igual que en el caso anterior, la aplicación web es una tienda que tiene dos vistas, en una se ve el catálogo de productos y en la otra el detalle de cada uno de los productos.

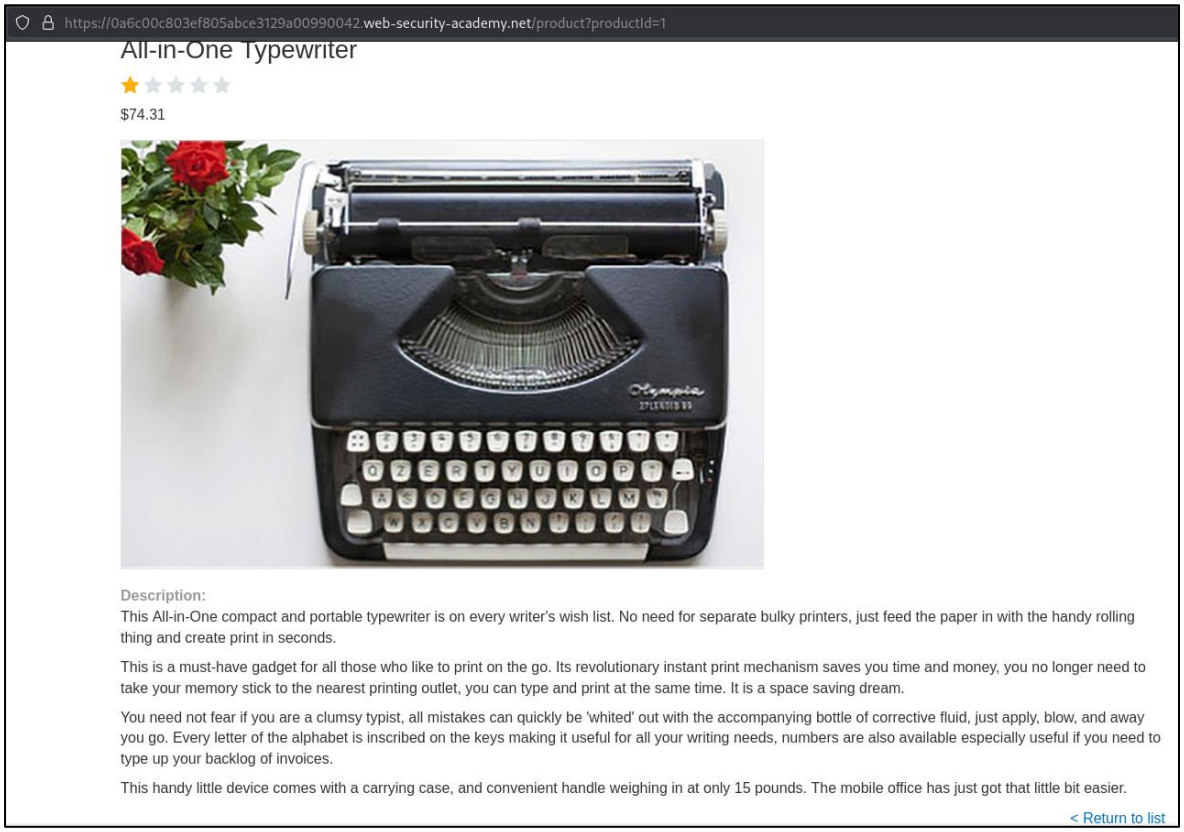

Ilustración 12: Descripción de producto

Inspeccionando el código se identifica la ruta y el formato de la petición que carga las imágenes.

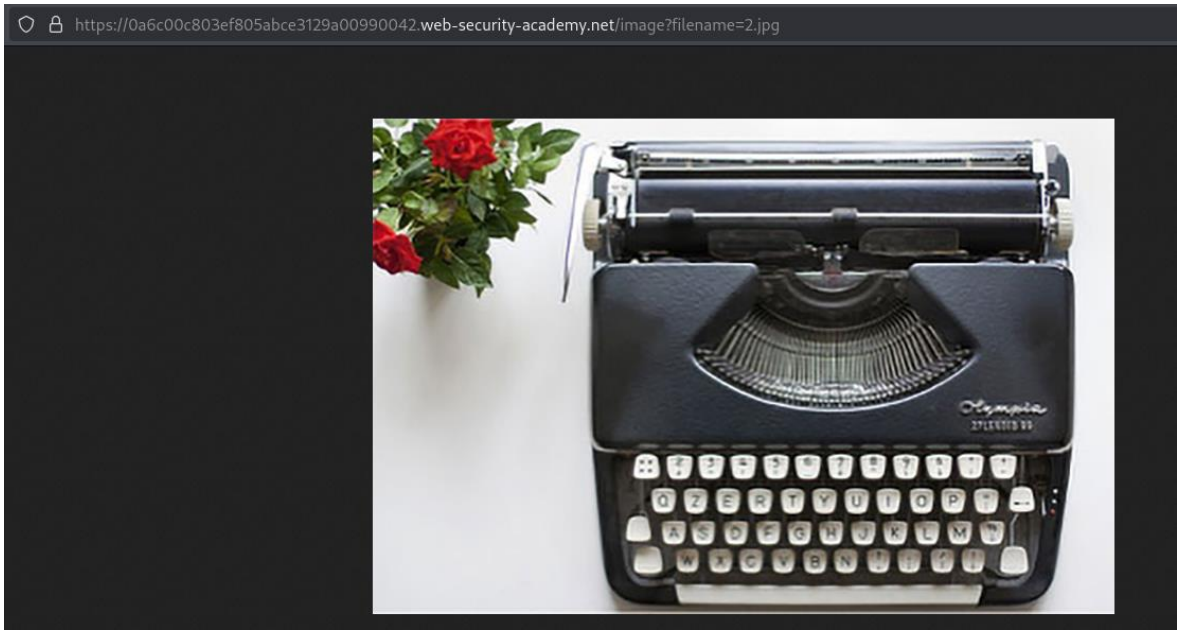

Ilustración 13: Petición que devuelve imágenes en el servidor

En este punto es el momento de observar la petición y la respuesta en el *proxy* web y ver en crudo que información se está pasando entre el cliente y el servidor. Repitiendo el proceso explicado en el caso anterior, se puede capturar la petición y observar la respuesta.

Cuando se realiza la petición a la `imagen 2.jpg` se puede apreciar que la respuesta es la imagen en crudo.

Ilustración 14: Respuesta que devuelve imagen en crudo

Teniendo identificado el punto en el que vamos a probar las inyecciones, al igual que en el caso anterior, es el momento de probar diferentes cargas útiles. En este caso, al estar implementadas diferentes medidas de seguridad se puede observar cómo ante diferentes *payloads* se obtiene una respuesta que indica que no se encuentra el archivo.

Al probar a realizar la petición con diferentes valores para el atributo `filename` como puede ser `'../../../../../../etc/passwd'` o bien

‘`../../../../../../../etc/passwd.jpg`’, se obtiene la misma respuesta por parte del servidor.

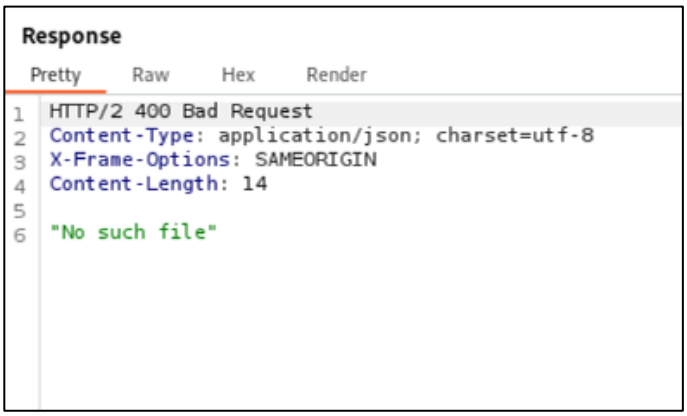

Ilustración 15: Respuesta del servidor de no se encuentra archivo

Como se ha comentado en la Sección 3 de este estudio, una de las pruebas para verificar el control de las rutas que recibe una aplicación, es el truncamiento mediante el uso de caracteres nulos. Probando a añadir en la carga útil un carácter nulo (`%00`) antes de la extensión de ficheros, se puede evadir la medida de seguridad que comprueba el valor de `filename`.

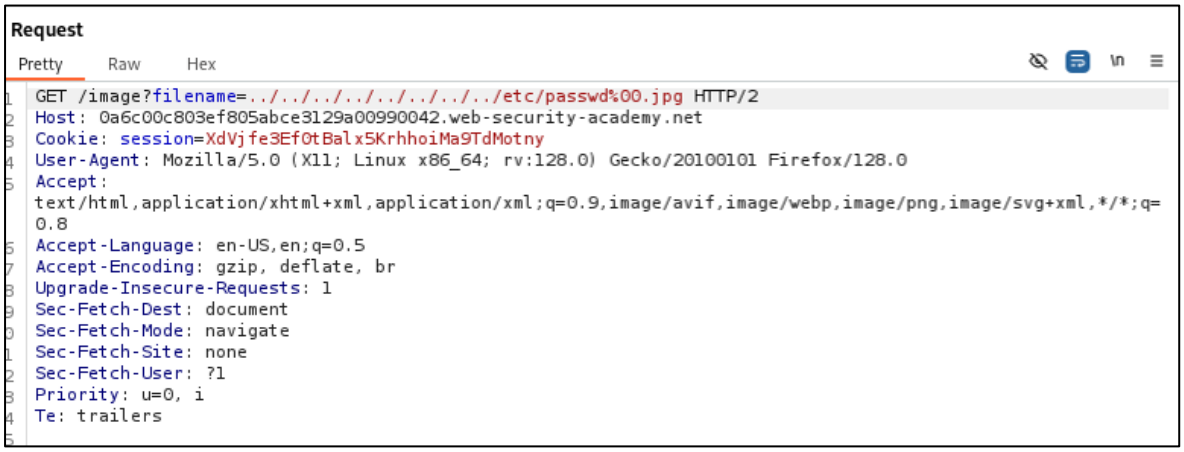

Ilustración 16: Evasión de medida de seguridad con el uso del carácter nulo

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

De este modo, el servidor devolverá el contenido del archivo encontrado en la ruta.

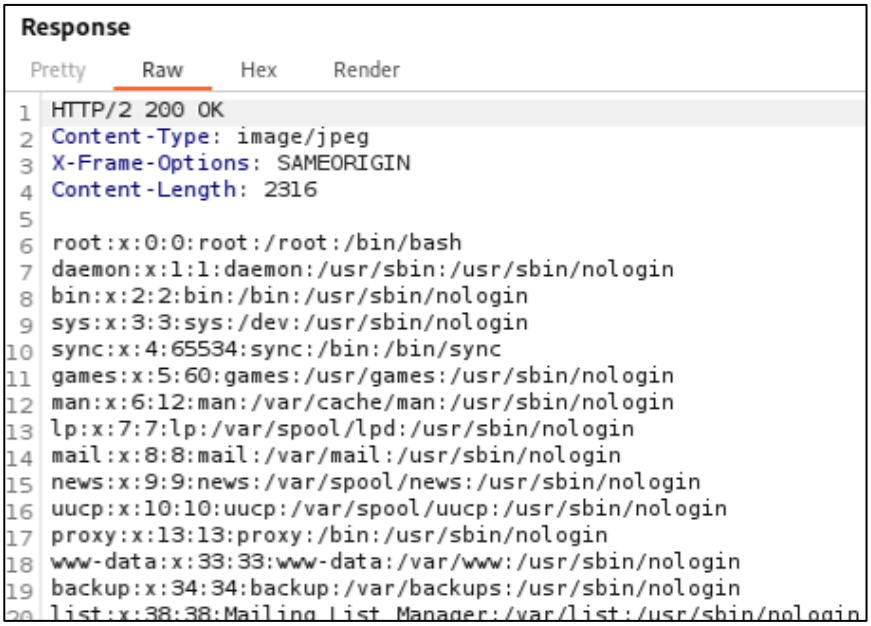

Ilustración 17: Contenido del archivo passwd del servidor

Con este ejemplo se puede apreciar como el uso de ciertos caracteres especiales puede ser usado para evadir medidas de protección.

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

6. Prevención y buenas prácticas

Prevenir la vulnerabilidad de *Path Traversal* es crucial para proteger la confidencialidad, integridad y disponibilidad de los sistemas. Un ataque exitoso puede permitir el acceso no autorizado a archivos sensibles, la modificación de configuraciones críticas, la escalada de privilegios y la exfiltración de datos, así como comprometer el cumplimiento normativo. Implementar medidas como la validación estricta de entradas, el uso de listas blancas y una configuración segura del servidor es esencial para mitigar estos riesgos y garantizar la seguridad del sistema.

6.1. Uso de listas blancas (*whitelisting*)

Es recomendable el uso de listas blancas en lugar de listas negras, si es posible hacerlo. Esto está justificado debido a que mientras las listas negras bloquean o filtran patrones que son considerados peligrosos, como secuencias de escape (`../`), es difícil anticipar todos los patrones maliciosos posibles, lo que puede dejar brechas de seguridad. Por ejemplo, algunos caracteres especiales pueden permitir a un atacante evadir una lista negra mediante codificación o variantes no contempladas del ataque.

Por el contrario, las listas blancas permiten únicamente valores que se sabe que son seguros y confiables, reduciendo considerablemente las posibilidades de explotación. Con esta estrategia, en lugar de intentar identificar todas las posibles entradas maliciosas, la validación se centra en permitir solo lo que es estrictamente necesario para la funcionalidad de la aplicación.

En el siguiente ejemplo, se implementa el uso de listas blancas para controlar qué archivos son los permitidos y que pueden ser accesibles a través de una aplicación. A continuación, se explica el código y su importancia:

```
$permitidos = array('doc1.txt', 'doc2.txt', 'manual.pdf');   // Lista blanca de archivos permitidos
$documento = $_GET['doc'];  // El nombre del archivo que solicita el usuario


// Verifica si el archivo solicitado está en la lista blanca
if (!in_array($documento, $permitidos)) {
    die('Archivo no permitido');    // Si el archivo no está permitido, detiene la ejecución
}
```

- **Lista blanca de archivos permitidos**: Se define un array llamado `$permitidos`, este contiene una lista de nombres de archivos que se consideran seguros y a los que los usuarios pueden acceder. En este caso, los archivos permitidos son '`doc1.txt`', '`doc2.txt`', y '`manual.pdf`'.
- **Entrada del usuario**: La variable `$documento` obtiene el valor del archivo solicitado a través del parámetro `doc` en la URL, utilizando `$_GET['doc']`. Este es el nombre del archivo que el usuario intenta acceder.

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

- **Verificación con la lista blanca**: La función `in_array()` se usa para verificar si el nombre del archivo que solicitó el usuario (`$documento`) está en la lista blanca (`$permitidos`).
  - Si el archivo no está en la lista blanca, la ejecución se detiene con el mensaje '*Archivo no permitido*'.
  - Si el archivo solicitado está en la lista blanca, el programa continuará su ejecución.

6.2. Restringir a directorios seguros

Se debe asegurar que el archivo solicitado por el usuario siempre se encuentre en un directorio controlado, verificando que la ruta del archivo nunca escape del directorio designado. Esto puede hacerse comparando la ruta absoluta del archivo con el directorio permitido.

El siguiente ejemplo, la técnica consiste en asegurarse de que el archivo solicitado esté ubicado dentro de un directorio controlado y que la ruta del archivo no escape del directorio permitido.

```
$documento = $_GET['doc'];  // Archivo solicitado por el usuario
$filepath = realpath('/var/www/html/documentos/' . $documento);
// Ruta absoluta del archivo


// Verifica que el archivo esté dentro del directorio permitido
if (strpos($filepath, '/var/www/html/documentos/') !== 0) {
    die('Acceso no permitido');    // Si no está dentro del
directorio, se detiene la ejecución
}
```

- **Solicitud del archivo por el usuario**:
  - La variable `$documento` obtiene el valor del nombre del archivo que el usuario está intentando acceder a través del parámetro `doc` en la URL (por ejemplo, `$_GET['doc']`).
- **Generar la ruta absoluta**:
  - La función `realpath()` convierte la ruta relativa del archivo solicitado en una ruta absoluta. Esto significa que, independientemente de las manipulaciones que intente el usuario (por ejemplo, usando secuencias como `../` para moverse a otros directorios), la función devolverá la ruta completa y segura del archivo en el sistema.
  - En este caso, se combina la ruta base del directorio (`/var/www/html/documentos/`) con el archivo solicitado (`$documento`) para generar la ruta completa.
- **Verificación de la ruta**:
  - La función `strpos()` se utiliza para verificar si la ruta absoluta del archivo (`$filepath`) empieza con el directorio permitido (`/var/www/html/documentos/`).

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

- Si la ruta absoluta no comienza con el directorio permitido, significa que el archivo solicitado está fuera del directorio seguro, por lo que el acceso es denegado.
- En este caso, se detiene la ejecución con el mensaje '*Acceso no permitido*'.

Para la mitigación del ejemplo expuesto imaginamos que el usuario intenta acceder a un archivo fuera del directorio permitido usando una ruta como:

```
http://ejemplo.com/script.php?doc=../../etc/passwd
```

En este caso, la función `realpath()` convertirá '`/var/www/html/documentos/../../etc/passwd`' en la ruta absoluta '`/etc/passwd`'. Al comparar la ruta con '`/var/www/html/documentos/`', se detectará que no está dentro del directorio permitido, y el acceso será denegado.

6.3. Verificación del formato

En lugar de permitir cualquier cadena como entrada, valida que el nombre del archivo tenga un formato esperado. Por ejemplo, se podría asegurar de que solo se permiten nombres de archivos sin rutas (`/`, `..`, etc.) o extensiones no peligrosas.

Al restringir el formato del nombre del archivo a una estructura segura, se previene que el usuario introduzca rutas relativas (como `../`) o caracteres peligrosos que podrían permitir escapar de los directorios controlados. Por otra parte, la limitación de extensiones peligrosas también ayuda a limitar las extensiones de archivos permitidas.

A continuación, el siguiente ejemplo muestra cómo implementar la verificación del formato para prevenir esta vulnerabilidad al validar que el nombre del archivo siga un formato seguro y esperado. Se utiliza una expresión regular para garantizar que el nombre del archivo cumpla ciertos criterios de seguridad.

```
$documento = $_GET['doc'];  // Nombre del archivo solicitado por el usuario


// Verifica que el nombre del archivo siga un formato seguro
if (!preg_match('/^[a-zA-Z0-9_-]+\.txt$/', $documento)) {
    die('Formato de archivo no permitido');  // Si el formato es incorrecto, se detiene la ejecución
}
```

- **Solicitud del archivo por el usuario**:
  - La variable `$documento` obtiene el valor del nombre del archivo que el usuario está intentando acceder a través del parámetro `doc` en la URL.
- **Verificación con expresión regular**:
  - La función `preg_match()` utiliza una expresión regular para verificar si el nombre del archivo tiene el formato correcto. En este caso, la expresión regular '`/^[a-zA-Z0-9_-]+\.txt$/`' asegura que el nombre del archivo comience y termine con letras (a-z, A-Z), números (0-9), guiones bajos (_), o

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

guiones medios (-) y, además, debe tener la extensión *.txt*, que es el único formato permitido en este caso.

- Si el nombre del archivo no coincide con este formato, el acceso es denegado y la ejecución se detiene con el mensaje '*Formato de archivo no permitido*'.

6.4. Sanitización de la entrada

La sanitización se refiere al proceso de limpiar o modificar los datos de entrada para eliminar caracteres o secuencias peligrosas, como las secuencias de '`../`' utilizadas en los ataques de *Path Traversal*. Este proceso puede evitar que los atacantes manipulen la entrada para acceder a archivos fuera del directorio permitido.

Entre las buenas prácticas de sanitización de entradas, son relevantes aquellas como:

- **Eliminación de secuencias de directorios ascendentes**: si por alguna razón no se puede restringir la entrada usando listas blancas, una forma de mitigación es eliminar las secuencias peligrosas como '`../`' o '`..\\`' de la entrada.

```
$documento = str_replace(array('../', '..\\'), '', $documento);
```

  - Función `str_replace`: busca en la cadena `$documento` las secuencias '`../`' y '`..\\`', que son comunes en ataques de *Path Traversal*. Estas secuencias permiten a un atacante 'subir' en la estructura de directorios (por ejemplo, salir del directorio actual y acceder a archivos ubicados en niveles superiores).
  - Eliminación de secuencias ascendentes: el código elimina todas las apariciones de secuencias de escape dentro de la variable *$*`documento`. Esto significa que, si un atacante intenta incluir estas secuencias en la entrada para acceder a archivos sensibles fuera del directorio designado, serán eliminadas, impidiendo la explotación.

- **Escapado de caracteres especiales**: dependiendo del entorno y del lenguaje de programación utilizado, puede ser útil escapar caracteres que podrían ser interpretados de forma maliciosa. Por ejemplo, en sistemas Unix, podrías asegurarte de que no se usen caracteres como el nulo en las rutas de archivos.

```
$filepath = realpath('/var/www/html/documentos/' . basename($documento));
```

  - Función `basename($documento)`: toma un nombre de archivo o ruta completa y devuelve solo el nombre del archivo, eliminando cualquier ruta que pueda haber sido proporcionada por el usuario. Esto evita que un atacante pueda usar secuencias como '`../`' o '`/`' para intentar escapar del directorio permitido, de manera que, si $documento tiene el valor `../../etc/passwd`, la función `basename($documento)` devolverá solo `passwd`, eliminando las secuencias de directorios ascendentes.
  - Función `realpath($filepath)`: convierte una ruta relativa en una ruta absoluta. También normaliza la ruta y elimina cualquier secuencia de directorios ascendentes como '`../`', garantizando que la ruta generada es la verdadera ubicación física del archivo en el sistema y que no contiene componentes que podrían ser manipulados por un atacante para acceder a archivos fuera del directorio permitido. Si el archivo no existe o si la ruta es maliciosa, `realpath()` devolverá false, lo que se puede usar para gestionar errores o impedir accesos.

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

6.5. Configuraciones seguras del servidor web

Configurar el servidor web para limitar el acceso a directorios críticos. Esto se puede hacer utilizando reglas de configuración en servidores como Apache o NGINX. En la siguiente configuración, podemos valorar como se declaran los siguientes valores:

- ***Options -Indexes***: desactiva el listado de directorios, lo que previene que los atacantes vean la estructura completa del sistema de archivos si intentan acceder a directorios no protegidos.
- ***Deny from all***: restringe el acceso a directorios específicos como `/private`.

```
<Directory /var/www/html/docs>
    Options -Indexes
    AllowOverride None
    Require all granted
</Directory>

<Directory /var/www/html/private>
    Deny from all
</Directory>
```

6.6. Monitorización y registro

La monitorización y registro del acceso a archivos es una mitigación clave para detectar y responder a posibles ataques de *Path Traversal*. Al implementar un sistema de monitoreo eficaz como un SIEM, las organizaciones pueden identificar comportamientos sospechosos que pueden indicar un intento de explotación de vulnerabilidades, permitiendo una respuesta oportuna para prevenir daños. La importancia de estas medidas mejora:

- **Detección temprana de ataques** en tiempo real y tomar medidas antes de que se produzcan daños significativos.
- **Responsabilidad y cumplimiento**, ayudando a las organizaciones a cumplir con normativas de protección de datos.
- **Mejora continua** al analizar los registros de acceso y los patrones de comportamiento, minimizando el riesgo de futuras vulnerabilidades.

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

7. Conclusión

La vulnerabilidad de *Path Traversal* es un riesgo crítico en la ciberseguridad que permite a los atacantes eludir las restricciones de acceso y acceder a archivos sensibles en un sistema. Este tipo de explotación puede comprometer la **integridad y confidencialidad** de datos vitales, lo que puede resultar en daños significativos para la organización, incluyendo la pérdida de información crítica y la posibilidad de ataques posteriores más graves. Por lo tanto, es esencial que las organizaciones tomen medidas proactivas para mitigar esta vulnerabilidad y proteger sus activos digitales.

La importancia de prevenir la vulnerabilidad de *Path Traversal* en las organizaciones no puede subestimarse y deben asegurar que sus sistemas y datos estén protegidos contra accesos no autorizados. Un solo incidente puede tener repercusiones graves, como el **robo de información sensible, daños a la reputación y consecuencias legales**. Por lo tanto, establecer una cultura de seguridad sólida y concienciar a los empleados sobre la importancia es fundamental para minimizar riesgos.

Desarrollar y mantener políticas efectivas contra la amenaza de *Path Traversal* es crucial para proteger los sistemas y datos sensibles. Estas políticas deben centrarse en prácticas de validación y sanitización de entradas, garantizando que cualquier dato ingresado por el usuario sea examinado cuidadosamente para detectar secuencias de navegación maliciosas y caracteres inusuales. La validación de entradas es esencial para que solo se permitan valores seguros, y el uso de listas blancas ayuda a restringir las rutas de acceso permitidas a aquellas específicamente aprobadas por la aplicación. Además, es importante incluir restricciones de acceso para que la aplicación solo interactúe con directorios seguros, reduciendo la posibilidad de que un atacante escape del entorno autorizado. Monitorización y registro de accesos también son componentes esenciales, ya que permiten a la organización detectar y rastrear comportamientos inusuales o intentos de acceso a archivos no autorizados, facilitando así la identificación y respuesta rápida a posibles incidentes.

Las **normativas y regulaciones relacionadas con la protección de datos**, como el RGPD o la HIPAA, así como la llegada de la reciente NIS2 subrayan la necesidad de proteger los sistemas contra vulnerabilidades de cualquier índole. Cumplir con estas normativas no solo es un requisito legal, sino que también ayuda a las organizaciones a establecer un marco para la gestión de riesgos y la protección de datos sensibles. Cumplir con estas normativas, ayuda a que las organizaciones puedan demostrar su compromiso con la seguridad y la privacidad de la información, lo que a su vez fortalece la confianza de sus clientes y socios comerciales.

Implementar estas prácticas mejora la seguridad general del sistema y fortalece la resiliencia de la organización frente a ataques de *Path Traversal*, haciendo que los futuros intentos de explotación sean menos probables o menos efectivos.

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

8. Acrónimos

- **API**: Application Programming Interface
- **ASCII**: American Standard Code for Information Interchange
- **CISA**: Cybersecurity and Infrastructure Security Agency
- **CERT**: Computer Emergency Response Team
- **CVE**: Common Vulnerabilities and Exposures
- **CWE**: Common Weakness Enumeration
- **CVSS**: Common Vulnerability Scoring System
- **DAST**: Dynamic Application Security Testing
- **DoS**: Denial of Service
- **HIPAA**: Health Insurance Portability and Accountability Act
- **HTTP**: Hypertext Transfer Protocol
- **HTTPS**: Hypertext Transfer Protocol Secure
- **IA**: Inteligencia Artificial
- **IDE**: Integrated Development Environment
- **IDS**: Intrusion Detection System
- **IPS**: Intrusion Prevention System
- **JSON**: JavaScript Object Notation
- **LLM**: Large Language Model
- **LFI**: Local File Inclusion
- **MITRE**: The MITRE Corporation
- **NIS2**: Network and Information Systems Directive 2
- **NIST**: National Institute of Standards and Technology
- **NVD**: National Vulnerability Database
- **OWASP**: Open Worldwide Application Security Project
- **PHP**: Hypertext Preprocessor
- **PoC**: Proof of Concept
- **RCE**: Remote Code Execution
- **REST**: Representational State Transfer
- **RFI**: Remote File Inclusion
- **RGPD**: Reglamento General de Protección de Datos
- **SAST**: Static Application Security Testing
- **SIEM**: Security Information and Event Management
- **SQL**: Structured Query Language
- **URL**: Uniform Resource Locator
- **URI**: Uniform Resource Identifier
- **UTF-8**: Unicode Transformation Format – 8 bits
- **WAF**: Web Application Firewall
- **XML**: eXtensible Markup Language
- **ZIP**: ZIP Archive File Format

9. Bibliografía

Referencia	Título, autor, y enlace web
[Ref.- 1]	*Path Traversal*, OWASP, https://owasp.org/www-community/attacks/*Path*_Traversal
[Ref.- 2]	*Testing Directory Traversal File Include*, OWASP, https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/01-Testing_Directory_Traversal_File_Include
[Ref.- 3]	*Path Traversal*, Portswigger Web Security Academy, https://portswigger.net/web-security/file-*path*-traversal
[Ref.- 4]	*File Inclusion/Path Traversal*, Hacktricks, https://book.hacktricks.xyz/pentesting-web/file-inclusion
[Ref.- 5]	*Directory Traversal*, Imperva, https://www.imperva.com/learn/application-security/directory-traversal/
[Ref.- 6]	*Comprehensive Guide on Path Traversal*, HackingArticles, https://www.hackingarticles.in/comprehensive-guide-on-*path*-traversal/
[Ref.- 7]	*Directory traversal*, Snyk, https://learn.snyk.io/lesson/directory-traversal/
[Ref.- 8]	*#01: Path Traversal*, Medium, https://medium.com/@karimelsayed0x1/01-*path*-traversal-0c52daffd26e
[Ref.- 9]	*CWE-22 – Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'),* MITRE, https://cwe.mitre.org/data/definitions/22.html
[Ref.- 10]	*Zip Slip Vulnerability***.** Snyk Security Research https://security.snyk.io/research/zip-slip-vulnerability

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