Contents

1 SEC-007: Traversal de Directorios

1.1 Severidad

**Alta **

1.2 Referencia OWASP

A01:2021 - Control de Acceso Roto

1.3 Referencias CWE

- CWE-22: Limitación Impropia de una Ruta a un Directorio Restringido (‘Path Traversal’)
- CWE-23: Traversal de Ruta Relativa

1.4 Descripción

Las vulnerabilidades de traversal de directorios ocurren cuando una aplicación usa entrada suministrada por usuarios para construir rutas de archivo sin la validación y sanitización adecuadas. Esto permite a los atacantes acceder a archivos y directorios fuera de la estructura de directorio prevista, potencialmente leyendo archivos sensibles (archivos de configuración, claves privadas, código fuente), modificando o eliminando archivos, o ejecutando código arbitrario. En servidores MCP que manejan operaciones de archivo, el traversal de directorios puede llevar al compromiso completo del sistema y exfiltración de datos.

1.5 Mapeo de Cumplimiento

1.5.1 SOC2

- **CC6.1**: Controles de Acceso Lógico y Físico - La organización restringe el acceso al sistema de archivos solo a recursos autorizados.

- **CC6.7**: Clasificación de Datos - Los archivos sensibles están protegidos del acceso no autorizado a través de controles de acceso.
- **CC7.2**: Monitoreo del Sistema - La organización monitorea patrones de acceso a archivos y detecta actividad sospechosa.

1.5.2 HIPAA

- **§164.312(a)(1)**: Control de Acceso - Implementar políticas técnicas para restringir el acceso a archivos y directorios de ePHI.
- **§164.312(c)(1)**: Controles de Integridad - Proteger archivos ePHI del acceso no autorizado mediante manejo seguro de archivos.
- **§164.312(c)(2)**: Mecanismo para Autenticar ePHI - Verificar acceso autorizado a ubicaciones de archivos ePHI.

1.5.3 PCI DSS

- **6.5.8**: Las vulnerabilidades de traversal de directorios se previenen mediante codificación segura
- **2.2.4**: Configurar parámetros de seguridad del sistema para prevenir acceso no autorizado a archivos
- **6.2.4**: Todo el software personalizado se desarrolla de forma segura con manejo adecuado de archivos
- **11.3.1**: Las vulnerabilidades de traversal de directorios se identifican mediante escaneo de vulnerabilidades

1.6 Ejemplo de Código Vulnerable

```
// □ INSEGURO: Construcción directa de ruta de archivo con entrada de usuario
const express = require('express');
const fs = require('fs');
const path = require('path');
const app = express();

const UPLOAD_DIR = '/uploads';

// Endpoint vulnerable: descarga de archivo con traversal de directorio
app.get('/api/download/:filename', (req, res) => {
  const filename = req.params.filename;

  // PELIGRO: Sin validación de ruta - el atacante puede usar ../../../
  // Entrada del atacante: "../../etc/passwd"
  const filepath = path.join(UPLOAD_DIR, filename);

  fs.readFile(filepath, (err, data) => {
    if (err) {
      return res.status(404).json({ error: 'Archivo no encontrado' });
    }
```

```
    res.download(data, filename);
  });
});

// Endpoint vulnerable: vista previa de archivo
app.post('/api/preview-file', (req, res) => {
  const { filename } = req.body;

  // PELIGRO: Concatenación directa de cadenas
  // Entrada del atacante: "../../../../etc/passwd"
  const filepath = UPLOAD_DIR + '/' + filename;

  fs.readFile(filepath, 'utf8', (err, data) => {
    if (err) {
      return res.status(404).json({ error: 'No encontrado' });
    }
    res.json({ preview: data.substring(0, 500) });
  });
});

// Endpoint vulnerable: eliminación de archivo
app.delete('/api/file/:id', (req, res) => {
  const { id } = req.params;

  // PELIGRO: Sin validación de ruta de archivo
  // El atacante puede eliminar archivos arbitrarios
  const filepath = `/var/data/${id}.json`;

  fs.unlink(filepath, (err) => {
    if (err) {
      return res.status(500).json({ error: 'Eliminación fallida' });
    }
    res.json({ success: true });
  });
});

app.listen(3000);
```

1.7 Implementación Segura

```
// ✅ SEGURO: Validación estricta de ruta de archivo y restricciones
const express = require('express');
const fs = require('fs');
const path = require('path');
const crypto = require('crypto');
const app = express();
```

```
const UPLOAD_DIR = path.resolve('/uploads');
const ALLOWED_EXTENSIONS = ['pdf', 'txt', 'jpg', 'png', 'json'];

// Función auxiliar para resolver ruta de archivo de forma segura
function resolveSafePath(baseDir, userInput) {
  // Validar formato de entrada
  if (!userInput || typeof userInput !== 'string' || userInput.length > 255) {
    throw new Error('Nombre de archivo inválido');
  }

  // Rechazar patrones peligrosos
  if (userInput.includes('..') || userInput.includes('\\') || userInput.startsWith
    throw new Error('Formato de nombre de archivo inválido');
  }

  // Obtener solo el nombre base del archivo (eliminar componentes de ruta)
  const basename = path.basename(userInput);

  // Verificar que la ruta resuelta está dentro del directorio base
  const resolvedPath = path.resolve(baseDir, basename);
  const resolvedBase = path.resolve(baseDir);

  if (!resolvedPath.startsWith(resolvedBase)) {
    throw new Error('Intento de traversal de directorio detectado');
  }

  return resolvedPath;
}

// Función auxiliar para validar extensión de archivo
function validateExtension(filename) {
  const ext = path.extname(filename).substring(1).toLowerCase();
  if (!ALLOWED_EXTENSIONS.includes(ext)) {
    throw new Error(`Tipo de archivo .${ext} no permitido`);
  }
  return ext;
}

// Endpoint seguro: descarga de archivo con validación de ruta
app.get('/api/download/:filename', (req, res) => {
  try {
    const filename = req.params.filename;

    // Validar extensión
    validateExtension(filename);

    // Resolver ruta de forma segura
    const filepath = resolveSafePath(UPLOAD_DIR, filename);
```

```
    // Verificación adicional: verificar que el archivo existe y es legible
    fs.accessSync(filepath, fs.constants.R_OK);

    // Verificar tamaño de archivo para prevenir lecturas de archivos grandes
    const stats = fs.statSync(filepath);
    if (stats.size > 100 * 1024 * 1024) { // máximo 100MB
      return res.status(400).json({ error: 'Archivo demasiado grande' });
    }

    // Verificar que es un archivo regular, no directorio o symlink
    if (!stats.isFile()) {
      return res.status(403).json({ error: 'Acceso denegado' });
    }

    res.download(filepath, filename);

  } catch (error) {
    console.error('Error de descarga:', error);
    res.status(400).json({ error: 'No se puede acceder al archivo' });
  }
});

// Endpoint seguro: vista previa de archivo con validación
app.post('/api/preview-file', (req, res) => {
  try {
    const { filename } = req.body;

    if (!filename || typeof filename !== 'string') {
      return res.status(400).json({ error: 'Nombre de archivo inválido' });
    }

    // Validar extensión
    validateExtension(filename);

    // Resolver ruta de forma segura
    const filepath = resolveSafePath(UPLOAD_DIR, filename);

    // Verificar accesibilidad del archivo
    fs.accessSync(filepath, fs.constants.R_OK);

    // Verificar tamaño de archivo
    const stats = fs.statSync(filepath);
    if (stats.size > 10 * 1024 * 1024) { // máximo 10MB para vista previa
      return res.status(400).json({ error: 'Archivo demasiado grande para vista pr
    }

    // Verificar que es un archivo regular
```

```
    if (!stats.isFile()) {
      return res.status(403).json({ error: 'Acceso denegado' });
    }

    // Leer con límite de tamaño
    const data = fs.readFileSync(filepath, 'utf8');
    const preview = data.substring(0, 1000);

    res.json({
      success: true,
      preview,
      size: stats.size
    });

  } catch (error) {
    console.error('Error de vista previa:', error);
    res.status(400).json({ error: 'No se puede acceder al archivo' });
  }
});

// Endpoint seguro: eliminación de archivo con lista blanca
app.delete('/api/file/:id', (req, res) => {
  try {
    const { id } = req.params;

    // Validar formato de ID (UUID o alfanumérico)
    if (!/^[a-zA-Z0-9-]{1,40}$/.test(id)) {
      return res.status(400).json({ error: 'ID de archivo inválido' });
    }

    // Construir nombre de archivo con ID validado
    const filename = `${id}.json`;

    // Resolver ruta de forma segura
    const filepath = resolveSafePath(UPLOAD_DIR, filename);

    // Verificar que el archivo existe antes de eliminar
    if (!fs.existsSync(filepath)) {
      return res.status(404).json({ error: 'Archivo no encontrado' });
    }

    // Verificar que es un archivo regular
    const stats = fs.statSync(filepath);
    if (!stats.isFile()) {
      return res.status(403).json({ error: 'No se puede eliminar este elemento' })
    }

    // Eliminar el archivo
```

```
    fs.unlinkSync(filepath);

    res.json({ success: true, message: 'Archivo eliminado' });

  } catch (error) {
    console.error('Error de eliminación:', error);
    res.status(500).json({ error: 'Eliminación fallida' });
  }
});

// Endpoint seguro: listar archivos en directorio
app.get('/api/files', (req, res) => {
  try {
    // Solo listar archivos en el directorio de carga
    const files = fs.readdirSync(UPLOAD_DIR);

    // Filtrar y validar cada archivo
    const safeFiles = files.filter(file => {
      try {
        // Solo incluir tipos de archivo permitidos
        const ext = path.extname(file).substring(1).toLowerCase();
        if (!ALLOWED_EXTENSIONS.includes(ext)) {
          return false;
        }

        // Verificar que es un archivo regular
        const filepath = path.join(UPLOAD_DIR, file);
        const stats = fs.statSync(filepath);
        return stats.isFile();
      } catch (error) {
        return false;
      }
    }).map(file => {
      const filepath = path.join(UPLOAD_DIR, file);
      const stats = fs.statSync(filepath);
      return {
        name: file,
        size: stats.size,
        modified: stats.mtime
      };
    });

    res.json({ files: safeFiles });

  } catch (error) {
    console.error('Error de listado:', error);
    res.status(500).json({ error: 'No se pueden listar archivos' });
  }
```

```
});

app.listen(3000);
```

1.8 Pasos de Remediación

1. **Usar Resolución de Ruta con Validación Estricta**: Usar `path.resolve()` para normalizar rutas y verificar que la ruta resuelta está dentro del directorio previsto. Rechazar cualquier entrada que contenga `..`, `/` o `\`. Usar `path.basename()` para extraer solo el nombre de archivo. Validar contra una lista blanca estricta de nombres de archivo o patrones permitidos. Implementar límites de longitud (255 caracteres para la mayoría de sistemas de archivos).
2. **Implementar Lista Blanca de Tipo de Archivo y Extensión**: Solo permitir extensiones de archivo específicas (.pdf, .txt, .json, etc.). Validar extensiones tanto en operaciones de carga como de descarga. Verificar que los tipos MIME coinciden con las extensiones de archivo. Almacenar archivos con un nombre basado en hash (p. ej., SHA256) en lugar de nombres suministrados por usuarios para eliminar completamente el riesgo de traversal de directorios.
3. **Usar Mapeo de ID de Archivo y Referencias Inmutables**: Almacenar archivos con identificadores únicos (UUID, hash) en lugar de nombres de archivo. Mantener una base de datos que mapee IDs de archivo a rutas de archivo reales. Esto previene cualquier manipulación directa de ruta y proporciona mejor control de acceso. Generar IDs de archivo del lado del servidor, nunca aceptarlos de usuarios.
4. **Implementar Defensa en Profundidad**: Verificar que los archivos existen y son archivos regulares (no directorios o symlinks) antes del acceso. Verificar tamaños de archivo para prevenir la lectura de archivos inesperadamente grandes. Usar principio de mínimo privilegio - ejecutar la aplicación con permisos mínimos del sistema de archivos. Implementar logging de acceso a archivos para auditoría. Usar características del sistema operativo (chroot, containerización) para restringir el alcance del sistema de archivos.

1.9 Referencias

- OWASP Path Traversal
- CWE-22: Traversal de Directorios
- OWASP Path Traversal Prevention Cheat Sheet
- PortSwigger: Path Traversal
- Node.js File System Security

Última Actualización: Enero 2026
Estado: Publicado
Idioma: Español