Reporte de Vulnerabilidad

Archivo: PeliculasController.cs

Código Analizado:

```
?using ApiPeliculas.Modelos;
using ApiPeliculas.Modelos.Dtos;
using ApiPeliculas.Repositorio.IRepositorio;
using Asp.Versioning;
using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace ApiPeliculas.Controllers.V1
{
    [Route("api/v{version:apiVersion}/peliculas")]
    [ApiController]
    [ApiVersion("1.0")]
    public class PeliculasController : ControllerBase
    {
        private readonly IPeliculaRepositorio _pelRepo;
        private readonly IMapper _mapper;

        public PeliculasController(IPeliculaRepositorio pelRepo, IMapper mapper)
        {
            _pelRepo = pelRepo;
            _mapper = mapper;
        }

        //V1
        //[AllowAnonymous]
        //[HttpGet]
        //[ProducesResponseType(StatusCodes.Status403Forbidden)]
        //[ProducesResponseType(StatusCodes.Status200OK)]
        //public IActionResult GetPeliculas()
        //{
        //    var listaPeliculas = _pelRepo.GetPeliculas();

        //    var listaPeliculasDto = new List<PeliculaDto>();

        //    foreach (var lista in listaPeliculas)
        //    {
        //        listaPeliculasDto.Add(_mapper.Map<PeliculaDto>(lista));
        //    }
        //    return Ok(listaPeliculasDto);
        //}

        //V2 con páginación
        [AllowAnonymous]
        [HttpGet]
        [ProducesResponseType(StatusCodes.Status403Forbidden)]
        [ProducesResponseType(StatusCodes.Status200OK)]
        public IActionResult GetPeliculas([FromQuery] int pageNumber = 1, [FromQuery] int
pageSize = 10)
        {
            try
            {
                var totalPeliculas = _pelRepo.GetTotalPeliculas();
                var peliculas = _pelRepo.GetPeliculas(pageNumber, pageSize);

                if (peliculas == null || !peliculas.Any())
                {
                    return NotFound("No se encontraron películas.");
                }

                var peliculasDto = peliculas.Select(p =>
_mapper.Map<PeliculaDto>(p)).ToList();

                var response = new
                {
                    PageNumber = pageNumber,
                    PageSize = pageSize,
                    TotalPages = (int)Math.Ceiling(totalPeliculas / (double)pageSize),
```

```
                    TotalItems = totalPeliculas,
                    Items = peliculasDto
                };

                return Ok(response);

            }
            catch (Exception)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "Error recuperando
datos de la aplicación");
            }
        }

        [AllowAnonymous]
        [HttpGet("{peliculaId:int}", Name = "GetPelicula")]
        [ProducesResponseType(StatusCodes.Status403Forbidden)]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        public IActionResult GetPelicula(int peliculaId)
        {
            var itemPelicula = _pelRepo.GetPelicula(peliculaId);

            if (itemPelicula == null)
            {
                return NotFound();
            }

            var itemPeliculaDto = _mapper.Map<PeliculaDto>(itemPelicula);

            return Ok(itemPeliculaDto);
        }

        //[Authorize(Roles = "Admin")]
        [HttpPost]
        [ProducesResponseType(201, Type = typeof(PeliculaDto))]
        [ProducesResponseType(StatusCodes.Status201Created)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        public IActionResult CrearPelicula([FromForm] CrearPeliculaDto crearPeliculaDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            if (crearPeliculaDto == null)
            {
                return BadRequest(ModelState);
            }

            if (_pelRepo.ExistePelicula(crearPeliculaDto.Nombre))
            {
                ModelState.AddModelError("", "La película ya existe");
                return StatusCode(404, ModelState);
            }

            var pelicula = _mapper.Map<Pelicula>(crearPeliculaDto);

            //if (!_pelRepo.CrearPelicula(pelicula))
            //{
            //    ModelState.AddModelError("", $"Algo salio mal guardando el
registro{pelicula.Nombre}");
            //    return StatusCode(404, ModelState);
            //}

            //Subida de Archivo
            if (crearPeliculaDto.Imagen != null)
            {
                string nombreArchivo = pelicula.Id + System.Guid.NewGuid().ToString() +
Path.GetExtension(crearPeliculaDto.Imagen.FileName);
                string rutaArchivo = @"wwwroot\ImagenesPeliculas\" + nombreArchivo;

                var ubicacionDirectorio = Path.Combine(Directory.GetCurrentDirectory(),
rutaArchivo);

                FileInfo file = new FileInfo(ubicacionDirectorio);
```

```
                if (file.Exists)
                {
                    file.Delete();
                }

                using (var fileStream = new FileStream(ubicacionDirectorio, FileMode.Create))
                {
                    crearPeliculaDto.Imagen.CopyTo(fileStream);
                }

                var baseUrl =
$"{HttpContext.Request.Scheme}://{HttpContext.Request.Host.Value}{HttpContext.Request.PathBase
.Value}";
                pelicula.RutaIMagen = baseUrl + "/ImagenesPeliculas/" + nombreArchivo;
                pelicula.RutaLocalIMagen = rutaArchivo;
            }
            else
            {
                pelicula.RutaIMagen = "https://placehold.co/600x400";
            }

            _pelRepo.CrearPelicula(pelicula);
            return CreatedAtRoute("GetPelicula", new { peliculaId = pelicula.Id }, pelicula);
        }

        //[Authorize(Roles = "Admin")]
        [HttpPatch("{peliculaId:int}", Name = "ActualizarPatchPelicula")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        public IActionResult ActualizarPatchPelicula(int peliculaId, [FromForm]
ActualizarPeliculaDto actualizarPeliculaDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            if (actualizarPeliculaDto == null || peliculaId != actualizarPeliculaDto.Id)
            {
                return BadRequest(ModelState);
            }

            var peliculaExistente = _pelRepo.GetPelicula(peliculaId);
            if (peliculaExistente == null)
            {
                return NotFound($"No se encontro la película con ID {peliculaId}");
            }

            var pelicula = _mapper.Map<Pelicula>(actualizarPeliculaDto);

            //if (!_pelRepo.ActualizarPelicula(pelicula))
            //{
            //    ModelState.AddModelError("", $"Algo salio mal actualizando el
registro{pelicula.Nombre}");
            //    return StatusCode(500, ModelState);
            //}

            //Subida de Archivo
            if (actualizarPeliculaDto.Imagen != null)
            {
                string nombreArchivo = pelicula.Id + System.Guid.NewGuid().ToString() +
Path.GetExtension(actualizarPeliculaDto.Imagen.FileName);
                string rutaArchivo = @"wwwroot\ImagenesPeliculas\" + nombreArchivo;

                var ubicacionDirectorio = Path.Combine(Directory.GetCurrentDirectory(),
rutaArchivo);

                FileInfo file = new FileInfo(ubicacionDirectorio);

                if (file.Exists)
                {
                    file.Delete();
                }

                using (var fileStream = new FileStream(ubicacionDirectorio, FileMode.Create))
                {
                    actualizarPeliculaDto.Imagen.CopyTo(fileStream);
                }
```

```
                var baseUrl =
$"{HttpContext.Request.Scheme}://{HttpContext.Request.Host.Value}{HttpContext.Request.PathBase
.Value}";
                pelicula.RutaIMagen = baseUrl + "/ImagenesPeliculas/" + nombreArchivo;
                pelicula.RutaLocalIMagen = rutaArchivo;
            }
            else
            {
                pelicula.RutaIMagen = "https://placehold.co/600x400";
            }

            _pelRepo.ActualizarPelicula(pelicula);
            return NoContent();

        }

        //[Authorize(Roles = "Admin")]
        [HttpDelete("{peliculaId:int}", Name = "BorrarPelicula")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        public IActionResult BorrarPelicula(int peliculaId)
        {

            if (!_pelRepo.ExistePelicula(peliculaId))
            {
                return NotFound();
            }

            var pelicula = _pelRepo.GetPelicula(peliculaId);

            if (!_pelRepo.BorrarPelicula(pelicula))
            {
                ModelState.AddModelError("", $"Algo salio mal borrando el
registro{pelicula.Nombre}");
                return StatusCode(500, ModelState);
            }

            return NoContent();
        }

        [AllowAnonymous]
        [HttpGet("GetPeliculasEnCategoria/{categoriaId:int}")]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        public IActionResult GetPeliculasEnCategoria(int categoriaId)
        {
            try
            {
                var listaPeliculas = _pelRepo.GetPeliculasEnCategoria(categoriaId);

                if (listaPeliculas == null || !listaPeliculas.Any())
                {
                    return NotFound($"No se encontraron películas en la categoría con ID
{categoriaId}.");
                }

                var itemPelicula = listaPeliculas.Select(pelicula =>
_mapper.Map<PeliculaDto>(pelicula)).ToList();
                //foreach (var pelicula in listaPeliculas)
                //{
                //    itemPelicula.Add(_mapper.Map<PeliculaDto>(pelicula));
                //}

                return Ok(itemPelicula);
            }
            catch (Exception)
            {

                return StatusCode(StatusCodes.Status500InternalServerError, "rror recuperando
datos de la aplicación");
            }

        }

        [AllowAnonymous]
        [HttpGet("Buscar")]
```

```
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        public IActionResult Buscar(string nombre)
        {
            try
            {
                var peliculas = _pelRepo.BuscarPelicula(nombre);
                if (!peliculas.Any())
                {
                    return NotFound($"No se encontraron películas que coincidan con los
criterios de búsqueda.");
                }

                var peliculasDto = _mapper.Map<IEnumerable<PeliculaDto>>(peliculas);
                return Ok(peliculasDto);
            }
            catch (Exception)
            {
                return StatusCode(StatusCodes.Status500InternalServerError, "Error recuperando
datos de la aplicación");
            }
        }
    }
}
```

Análisis: ```html
Análisis de Seguridad y Calidad del Código

Vulnerabilidades Identificadas

- **Vulnerabilidad:** Path Traversal / Local File Inclusion (LFI)
  Línea Aproximada: En los métodos `CrearPelicula` y `ActualizarPatchPelicula` (ej., línea 121 y 200 respectivamente).
  Descripción: El código construye rutas de archivo concatenando cadenas directamente con datos proporcionados por el usuario (nombre de archivo), lo que permite a un atacante manipular el nombre del archivo (`crearPeliculaDto.Imagen.FileName` o `actualizarPeliculaDto.Imagen.FileName`) para incluir rutas relativas como `../../` y acceder o sobrescribir archivos fuera del directorio previsto (`wwwroot\ImagenesPeliculas`). Además, la validación de la extensión es inexistente, lo que permite subir archivos ejecutables.
  Impacto: Un atacante podría leer archivos sensibles del servidor, ejecutar código arbitrario (si puede subir un archivo ejecutable y luego ejecutarlo), o sobrescribir archivos de configuración, comprometiendo la seguridad y la integridad del sistema.
  Métricas afectadas: Seguridad.
- **Vulnerabilidad:** Falta de validación y manejo de excepciones robusto en la eliminación de archivos.
  Línea Aproximada: En los métodos `CrearPelicula` y `ActualizarPatchPelicula` (ej., línea 128 y 207 respectivamente).
  Descripción: El código intenta eliminar archivos sin un manejo de excepciones adecuado. Si la eliminación falla (por permisos, archivo en uso, etc.), la aplicación no lo detecta ni lo maneja correctamente, lo que puede llevar a inconsistencias o fallos inesperados.
  Impacto: Potencial pérdida de datos, denegación de servicio (si la eliminación falla repetidamente y consume recursos), o inconsistencias en el estado del sistema.
  Métricas afectadas: Confiabilidad.
- **Vulnerabilidad:** Ausencia de autorización en endpoints de creación, actualización y eliminación.
  Línea Aproximada: Se comentan las directivas `[Authorize(Roles = "Admin")]` en `HttpPost`, `HttpPatch` y `HttpDelete`.
  Descripción: Al comentar las directivas de autorización, se permite que usuarios no autorizados accedan a funciones críticas de creación, actualización y eliminación de películas.
  Impacto: Usuarios no autorizados podrían modificar o eliminar datos, comprometiendo la integridad de la información y la funcionalidad del sistema.
  Métricas afectadas: Seguridad.

Métricas de Calidad del Código

- **Complejidad Ciclomática:** Moderada. Algunos métodos, como `CrearPelicula` y `ActualizarPatchPelicula`, tienen una complejidad alta debido a las múltiples ramas condicionales (if/else) relacionadas con la gestión de la imagen.
  Impacto: Dificultad para entender, mantener y probar el código.
- **Duplicación:** Alta. La lógica de manejo de archivos (subida, almacenamiento, y generación de URL) se duplica en los métodos `CrearPelicula` y `ActualizarPatchPelicula`.
  Impacto: Incremento del riesgo de errores y dificultad en el mantenimiento. Si se necesita cambiar la forma en que se manejan los archivos, se debe modificar en múltiples lugares.
- **Legibilidad:** Aceptable, pero puede mejorar. El código es generalmente fácil de entender, pero la presencia de lógica compleja y duplicada dificulta su comprensión.
  Impacto: Dificultad para que otros desarrolladores entiendan el código y colaboren.
- **Acoplamiento:** Moderado. La clase depende de `IPeliculaRepositorio`, `IMapper`, `IFormFile` (a través de los DTOs) y `HttpContext`.
  Impacto: Cambios en las dependencias podrían requerir modificaciones en la clase.

Solución Propuesta

Mitigación de Vulnerabilidades y Mejora de Calidad

- **Mitigación de Path Traversal/LFI:**
  - **Validar y limpiar el nombre del archivo:** Utilizar expresiones regulares o funciones de sanitización para eliminar caracteres no permitidos y evitar rutas relativas (`../`).
  - **Validar la extensión del archivo:** Asegurarse de que la extensión del archivo sea una extensión permitida (ej., `.jpg`, `.png`). Usar `Path.GetExtension()` y una lista blanca de extensiones permitidas.
  - **Utilizar `Path.Combine()` para construir rutas:** En lugar de concatenar cadenas directamente, usar `Path.Combine()` para asegurar que la ruta sea construida correctamente, independientemente del sistema operativo.
  - **Generar nombres de archivo aleatorios:** Evitar usar el nombre original del archivo proporcionado por el usuario. Generar un nombre aleatorio y almacenar la extensión original por separado si es necesario.
  - **Almacenar archivos fuera del directorio público (wwwroot):** Guardar los archivos en un directorio que no sea accesible directamente a través de la web. Crear un endpoint que sirva las imágenes, implementando la autorización y validación adecuadas.

  Ejemplo de código (parcial): ```csharp // Generar nombre de archivo seguro string extension = Path.GetExtension(crearPeliculaDto.Imagen.FileName); if (!EsExtensionPermitida(extension)) { return BadRequest("Extensión de archivo no permitida."); } string nombreArchivo = Guid.NewGuid().ToString() + extension; string rutaArchivo = Path.Combine(@"ImagenesPeliculas", nombreArchivo); //Guardar fuera del wwwroot // Validar y crear directorio si no existe string directorioBase = Path.Combine(Directory.GetCurrentDirectory(), "ImagenesPeliculas"); if (!Directory.Exists(directorioBase)) { Directory.CreateDirectory(directorioBase); } var ubicacionDirectorio = Path.Combine(directorioBase, nombreArchivo); using (var fileStream = new FileStream(ubicacionDirectorio, FileMode.Create)) { crearPeliculaDto.Imagen.CopyTo(fileStream); } pelicula.RutaIMagen = $"{HttpContext.Request.Scheme}://{HttpContext.Request.Host.Value}/api/imagenes/{nombreArchivo}"; //Endpoint controlado pelicula.RutaLocalIMagen = rutaArchivo; ```
- **Mejora en el manejo de excepciones en la eliminación de archivos:**
  - **Usar bloques `try-catch` para capturar excepciones:** Envolver el código de eliminación de archivos en un bloque `try-catch` para capturar excepciones como `IOException` o `UnauthorizedAccessException`.
  - **Registrar errores:** Usar un logger para registrar cualquier error que ocurra durante la eliminación de archivos.

    - **Implementar una estrategia de reintento:** Si la eliminación falla, intentar de nuevo después de un breve retraso.

  Ejemplo de código (parcial): ```csharp try { file.Delete(); } catch (IOException ex) { // Registrar el error _logger.LogError($"Error al eliminar el archivo: {ex.Message}"); // Manejar el error (ej., mostrar un mensaje al usuario) return StatusCode(StatusCodes.Status500InternalServerError, "Error al eliminar el archivo."); } ```

- **Implementar Autorización Adecuada:**
    - **Habilitar la autorización:** Descomentar y configurar correctamente los atributos `[Authorize(Roles = "Admin")]` en los métodos `CrearPelicula`, `ActualizarPatchPelicula` y `BorrarPelicula`.
    - **Validar roles:** Asegurarse de que los roles de usuario están correctamente configurados y asignados en el sistema de autenticación.

  Ejemplo de código: ```csharp [Authorize(Roles = "Admin")] [HttpPost] public IActionResult CrearPelicula([FromForm] CrearPeliculaDto crearPeliculaDto) { // ... } ```
- **Mejora de la Calidad del Código:**
    - **Refactorización:** Extraer la lógica de manejo de archivos en un método o clase separada para reducir la duplicación y mejorar la legibilidad. Aplicar el principio DRY (Don't Repeat Yourself).
    - **Reducción de la complejidad:** Dividir los métodos complejos en métodos más pequeños y fáciles de entender.
    - **Inyección de dependencias:** Utilizar la inyección de dependencias para facilitar las pruebas unitarias y reducir el acoplamiento. Por ejemplo, inyectar un servicio de manejo de archivos en lugar de instanciarlo directamente en la clase.
    - **Mejorar la legibilidad:** Utilizar nombres de variables y métodos descriptivos, agregar comentarios para explicar la lógica compleja, y seguir las convenciones de codificación del lenguaje.

```
```