Vulnerability Report

Archivo: PeliculasController.cs

Code Analyzed:

```
?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");
            }
        }
    }
}
```

Analysis: ```html

Security Vulnerabilities and Code Quality Analysis

Potential Security Vulnerabilities

Directory Traversal

Type: Directory Traversal/Path Manipulation

Line: Approximately lines 102 and 180.

Description: The code constructs file paths using string concatenation, which can be vulnerable to directory traversal attacks. A malicious user could manipulate the `crearPeliculaDto.Imagen.FileName` or `actualizarPeliculaDto.Imagen.FileName` to include directory traversal sequences (e.g., "../") to access or overwrite files outside of the intended directory. This could lead to arbitrary file read or write, potentially compromising the entire system.

Mitigation:

- Use `Path.GetFileName()` to extract only the file name from `crearPeliculaDto.Imagen.FileName` or `actualizarPeliculaDto.Imagen.FileName`.
- Sanitize the file name to remove or encode any potentially malicious characters.
- Use `Path.Combine()` to construct the full file path safely, ensuring that the base path is always respected.
- Implement proper input validation and file type validation.

Lack of Authorization

Type: Missing Authorization

Line: 69, 152, 169, 198

Description: The `HttpPost` `HttpPatch` and `HttpDelete` actions are commented out `//[Authorize(Roles = "Admin")]`. This means that they are not protected and can be accessed by anyone. This allows unauthorized users

to create, update, and delete movies.

Mitigation:

- Uncomment the `[Authorize(Roles = "Admin")]` attribute to restrict access to the actions.
- Consider a more robust authorization scheme using claims-based authorization or policy-based authorization.

Exception Handling: Potential Information Leakage

Type: Information Leakage Through Exception Handling

Line: Approximately lines 50, 227, 249.

Description: The catch blocks return a generic error message "Error recuperando datos de la aplicación". While this avoids exposing sensitive details directly to the user, more detailed logging can be extremely valuable for debugging and proactive problem resolution. If detailed error information isn't logged, it can make diagnosing and resolving issues in production difficult.

Mitigation:

- Implement detailed logging to capture the specific exception, stack trace, and relevant context information. Use a logging framework like Serilog or NLog to log to a persistent store (e.g., database, file, or dedicated logging service)
- Consider using custom exception types to provide more context about the error.
- Implement a global exception handler to catch unhandled exceptions and log them appropriately.

Missing input Validation on File Uploads

Type: Missing input Validation

Line: Approximately lines 90, 171

Description: There is no check on the `crearPeliculaDto.Imagen` object for file type, size or other constraints, this can cause issues such as uploading large or dangerous files.

Mitigation:

- Validate the file extension is of an expected type.
- Validate the file size is within limits.
- Check the `Content-Type` HTTP header to confirm that the uploaded file matches the expected type.

Code Quality Metrics

Duplication

Description: There is significant code duplication in the `CrearPelicula` and `ActualizarPatchPelicula` actions, particularly in the file upload logic. This violates the DRY (Don't Repeat Yourself) principle.

Improvement: Extract the file upload logic into a separate, reusable method. This will reduce code duplication and improve maintainability.

Readability

Description: The code is generally readable but could be improved by adding more comments to explain complex logic, especially in the file upload and exception handling sections.

Improvement: Add comments to clarify the purpose of each step in the file upload process and the handling of different error conditions.

Coupling

Description: The controller is tightly coupled to the `IPeliculaRepositorio`. While dependency injection is used, further decoupling could be achieved by introducing an additional abstraction layer, such as a service layer, to handle business logic.

Improvement: Introduce a service layer between the controller and the repository. This will encapsulate the business logic and reduce the controller's direct dependency on the repository.

Complexity

Description: The methods related to file handling (`CrearPelicula`, `ActualizarPatchPelicula`) exhibit moderate complexity due to the steps involved in file processing, directory operations, and exception handling.

Improvement: Decompose these larger methods into smaller, more focused functions, each responsible for a specific task such as validating the file, saving the file, or handling errors. Aim to simplify the control flow within each method by reducing the number of conditional statements and nested loops. Use well-named variables and consistent formatting to make the code easier to understand.

Proposed Solution

1. **Address Directory Traversal:** Implement `Path.GetFileName()` and `Path.Combine()` in the file upload logic. Sanitize file names to prevent malicious input.
2. **Implement Authorization:** Uncomment the `[Authorize(Roles = "Admin")]` attribute to restrict access to the create, update and delete actions.
3. **Improve Exception Handling:** Implement detailed logging with a framework like Serilog to capture exception details.
4. **Add input validation to File Uploads:** Implement file type validation and size limits.
5. **Reduce Code Duplication:** Extract the file upload logic into a separate, reusable method.
6. **Improve Readability:** Add more comments to explain complex logic.
7. **Reduce Coupling:** Introduce a service layer to handle business logic between the controller and the repository.
8. **Reduce Method Complexity:** Decompose complex methods related to file processing and database interactions into smaller, focused functions, each with a single responsibility.

```
```