Reporte de Vulnerabilidad

Archivo: CategoriasController.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.Cors;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace ApiPeliculas.Controllers.V1
{

    //[Route("api/[controller]")] //Opción estática
    //[Authorize(Roles = "Admin")]
    //[ResponseCache(Duration = 20)]
    [Route("api/v{version:apiVersion}/categorias")]
    [ApiController]
    [ApiVersion("1.0")]
    //[Obsolete("Esta versión del controlador está obsoleta")]
    public class CategoriasController : ControllerBase
    {
        private readonly ICategoriaRepositorio _ctRepo;
        private readonly IMapper _mapper;

        public CategoriasController(ICategoriaRepositorio ctRepo, IMapper mapper)
        {
            _ctRepo = ctRepo;
            _mapper = mapper;
        }

        [HttpGet("GetString")]
        [Obsolete("Este endpoint esta obsoleto, por favor use el endpoing de la versión 2.0")]
        //[MapToApiVersion("2.0")]
        public IEnumerable<string> Get()
        {
            return new string[] { "valor1", "valor2", "valor3" };
        }


        [AllowAnonymous]
        [HttpGet]
        //[MapToApiVersion("1.0")]
        //[ResponseCache(Duration = 20)]
        [ResponseCache(CacheProfileName = "PorDefecto30Segundos")]
        [ProducesResponseType(StatusCodes.Status403Forbidden)]
        [ProducesResponseType(StatusCodes.Status200OK)]
        //[EnableCors("PoliticaCors")] // Aplica la política CORS a este método
        public IActionResult GetCategorias()
        {
            var listaCategorias = _ctRepo.GetCategorias();

            var listaCategoriasDto = new List<CategoriaDto>();

            foreach (var lista in listaCategorias)
            {
                listaCategoriasDto.Add(_mapper.Map<CategoriaDto>(lista));
            }
            return Ok(listaCategoriasDto);
        }



        [AllowAnonymous]
        [HttpGet("{categoriaId:int}", Name = "GetCategoria")]
        //[ResponseCache(Duration = 40)]
        //[ResponseCache(Location = ResponseCacheLocation.None, NoStore = true)]
```

```
        [ResponseCache(CacheProfileName = "PorDefecto30Segundos")]
        [ProducesResponseType(StatusCodes.Status403Forbidden)]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        public IActionResult GetCategoria(int categoriaId)
        {
            var itemCategoria = _ctRepo.GetCategoria(categoriaId);

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

            var itemCategoriaDto = _mapper.Map<CategoriaDto>(itemCategoria);

            return Ok(itemCategoriaDto);
        }

        //[Authorize(Roles = "Admin")]
        [HttpPost]
        [ProducesResponseType(StatusCodes.Status201Created)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        public IActionResult CrearCategoria([FromBody] CrearCategoriaDto crearCategoriaDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

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

            if (_ctRepo.ExisteCategoria(crearCategoriaDto.Nombre))
            {
                ModelState.AddModelError("", "La categoría ya existe");
                return StatusCode(404, ModelState);
            }

            var categoria = _mapper.Map<Categoria>(crearCategoriaDto);

            if (!_ctRepo.CrearCategoria(categoria))
            {
                ModelState.AddModelError("", $"Algo salio mal guardando el
registro{categoria.Nombre}");
                return StatusCode(404, ModelState);
            }

            return CreatedAtRoute("GetCategoria", new { categoriaId = categoria.Id },
categoria);
        }

        //[Authorize(Roles = "Admin")]
        [HttpPatch("{categoriaId:int}", Name = "ActualizarPatchCategoria")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        public IActionResult ActualizarPatchCategoria(int categoriaId, [FromBody] CategoriaDto
categoriaDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            if (categoriaDto == null || categoriaId != categoriaDto.Id)
            {
                return BadRequest(ModelState);
            }

            var categoriaExistente = _ctRepo.GetCategoria(categoriaId);
            if (categoriaExistente == null)
            {
                return NotFound($"No se encontro la categoría con ID {categoriaId}");
            }
```

```
            var categoria = _mapper.Map<Categoria>(categoriaDto);

            if (!_ctRepo.ActualizarCategoria(categoria))
            {
                ModelState.AddModelError("", $"Algo salio mal actualizando el
registro{categoria.Nombre}");
                return StatusCode(500, ModelState);
            }

            return NoContent();
        }

        //[Authorize(Roles = "Admin")]
        [HttpPut("{categoriaId:int}", Name = "ActualizarPutCategoria")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        [ProducesResponseType(StatusCodes.Status404NotFound)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        public IActionResult ActualizarPutCategoria(int categoriaId, [FromBody] CategoriaDto
categoriaDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            if (categoriaDto == null || categoriaId != categoriaDto.Id)
            {
                return BadRequest(ModelState);
            }

            var categoriaExistente = _ctRepo.GetCategoria(categoriaId);
            if (categoriaExistente == null)
            {
                return NotFound($"No se encontro la categoría con ID {categoriaId}");
            }

            var categoria = _mapper.Map<Categoria>(categoriaDto);

            if (!_ctRepo.ActualizarCategoria(categoria))
            {
                ModelState.AddModelError("", $"Algo salio mal actualizando el
registro{categoria.Nombre}");
                return StatusCode(500, ModelState);
            }

            return NoContent();
        }

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

            if (!_ctRepo.ExisteCategoria(categoriaId))
            {
                return NotFound();
            }

            var categoria = _ctRepo.GetCategoria(categoriaId);

            if (!_ctRepo.BorrarCategoria(categoria))
            {
                ModelState.AddModelError("", $"Algo salio mal borrando el
registro{categoria.Nombre}");
                return StatusCode(500, ModelState);
            }

            return NoContent();
        }
    }
}
```

Análisis: ```html
Vulnerabilidades de Seguridad

Falta de Autorización Consistente

Tipo: Control de acceso inapropiado.
Líneas Afectadas: Comentarios `//[Authorize(Roles = "Admin")]` en varios métodos (HttpPost, HttpPatch, HttpPut, HttpDelete).
Descripción: Aunque hay autorización a nivel de controlador `//[Authorize(Roles = "Admin")]`, está comentada. Los métodos individuales, HttpPost, HttpPatch, HttpPut, HttpDelete, están protegidos solo si el comentario se elimina y se habilita la autorización, tal cual se ve ahora, no están protegidos. Esto permite que usuarios no autorizados creen, actualicen y eliminen categorías. La vulnerabilidad se agrava porque `[AllowAnonymous]` esta explícitamente habilitado para Get. Se debe definir una política clara sobre quién puede acceder y modificar los recursos.
Mitigación: Eliminar los comentarios `//` de los atributos `[Authorize(Roles = "Admin")]` para habilitar la restricción de acceso a los roles administrativos. Si se requiere granularidad, mantener la autorización a nivel de método. Validar roles en cada operación sensible.
Mejora: Centralizar la gestión de autorización, posiblemente usando políticas de autorización más complejas que permitan un control más fino sobre quién puede hacer qué.

Métricas de Calidad del Código

Duplicación de Código

Descripción: Existe duplicación de código en los métodos `ActualizarPatchCategoria` y `ActualizarPutCategoria`. Ambos realizan validaciones similares (ModelState, nulidad de DTO, existencia de la categoría) y lógica de actualización.
Mejora: Refactorizar la lógica común en una función privada o un servicio auxiliar reutilizable. Esto reduce la duplicación, mejora la legibilidad y facilita el mantenimiento. Por ejemplo, crear una función `ValidarCategoria` que contenga la lógica de validación y sea llamada por ambos métodos.

Complejidad Ciclomática

Descripción: Algunos métodos, especialmente los de actualización y creación, tienen una complejidad ciclomática moderada debido a múltiples validaciones y comprobaciones de errores.
Mejora: Extraer partes de la lógica en métodos separados para reducir la complejidad de cada método individual. Usar patrones de diseño como "Guard Clause" (cláusulas de guarda) para simplificar el flujo de control. Esto hará que el código sea más fácil de entender y probar.

Acoplamiento

Descripción: El controlador depende directamente del repositorio `ICategoriaRepositorio` y del mapper `IMapper`.
Mejora: Si el acoplamiento se vuelve un problema, considerar el uso de un patrón Mediator o una capa de servicio para desacoplar aún más el controlador de las dependencias directas del repositorio y el mapper. Esto facilitaría las pruebas unitarias y permitiría cambiar las implementaciones subyacentes sin afectar al controlador.

Legibilidad

Descripción: En general, el código es legible, pero hay margen de mejora en la consistencia del manejo de errores y en la claridad de los mensajes de error.
Mejora: Estandarizar el manejo de errores (por ejemplo, usar siempre `return BadRequest()` para errores de validación). Proporcionar mensajes de error más descriptivos y específicos para facilitar la depuración. Utilizar nombres de variables y métodos que sean autoexplicativos.

Manejo de Errores Inconsistente

Descripción: El manejo de errores no es consistente en todos los métodos. Algunos usan `return BadRequest(ModelState)`, otros usan `return StatusCode(404, ModelState)` o `return StatusCode(500, ModelState)`, y otros usan `return NotFound()`.
Mejora: Unificar el enfoque de manejo de errores. Considerar el uso de `ProblemDetails` para una respuesta de error más estandarizada y rica en información. Implementar un middleware de manejo de excepciones para capturar excepciones no controladas y devolver respuestas de error consistentes.

Solución Propuesta

1. **Habilitar Autorización:** Descomentar las líneas `[Authorize(Roles = "Admin")]` en los métodos que requieran autorización administrativa.
2. **Refactorizar Lógica Duplicada:** Crear métodos o servicios auxiliares para la lógica común en `ActualizarPatchCategoria` y `ActualizarPutCategoria`.
3. **Mejorar el Manejo de Errores:** Unificar el manejo de errores usando `ProblemDetails` y un middleware de manejo de excepciones.
4. **Simplificar la Complejidad:** Extraer la lógica compleja en métodos separados y usar "Guard Clauses".
5. **Estandarizar Mensajes de Error:** Proporcionar mensajes de error más descriptivos y específicos.

```
```