Vulnerability Report

Archivo: CategoriasController.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.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();
        }
    }
}
```

Analysis: ```html
Security Vulnerabilities
Vulnerability: Insecure Direct Object Reference (IDOR) - Patch/Put/Delete methods
Approximate Line: 134, 157, 185
Description: The `ActualizarPatchCategoria`, `ActualizarPutCategoria`, and `BorrarCategoria` methods rely on the `categoriaId` passed in the route and the `categoriaDto.Id` in the request body for identifying the category to update/delete. A malicious user could potentially modify the `categoriaDto.Id` to manipulate a different category than intended if proper authorization is not in place or if the correct entity is not validated from the repository using the user's context.
Mitigation: Ensure that the user has the authority to modify the specified category. Validate, within the repository layer, that the category actually belongs to the user attempting to modify it. Do not only compare Ids between DTO and parameter but also with the information stored in the database and user context.
Improvement: Employ robust authorization mechanisms and contextualize database lookups within the repository.
Vulnerability: Mass Assignment - Patch/Put methods
Approximate Line: 132, 155
Description: The `ActualizarPatchCategoria` and `ActualizarPutCategoria` methods directly map the `CategoriaDto` to the `Categoria` entity using AutoMapper. If the `CategoriaDto` contains properties that shouldn't be modified by users, this can lead to mass assignment vulnerabilities where malicious users can overwrite sensitive data.
Mitigation: Use a more specific DTO containing only the properties that are allowed to be updated. Consider using explicit property mappings instead of AutoMapper, or configure AutoMapper to ignore specific properties.
Improvement: Define separate DTOs for updating and retrieving categories to control which properties can be modified.
Vulnerability: Lack of Authorization
Approximate Line: 53, 69, 98, 121, 145, 172
Description: The comments suggest `Authorize(Roles = "Admin")` was used for some endpoints, but is now commented out. This may mean all authenticated users can access privileged actions like creating, updating, and deleting categories.
Mitigation: Re-enable the `[Authorize]` attribute with appropriate role-based or policy-based authorization on the `HttpPost`, `HttpPut`, `HttpPatch`, and `HttpDelete` actions. Consider using scopes or claims-based authorization for more granular control.
Improvement: Implement a robust authorization strategy and carefully review endpoint accessibility requirements.
Vulnerability: Inconsistent Error Handling
Approximate Line: 110, 138, 161, 189
Description: The error handling approach is inconsistent, sometimes returning a `StatusCode` with `ModelState` and sometimes returning a simple `NotFound` or `BadRequest`.
Mitigation: Establish a consistent error handling strategy throughout the controller and application. Use exception filters for centralized exception handling. Consider returning ProblemDetails responses for standardized error information.
Improvement: Implement a centralized exception handling mechanism for better maintainability and consistency.
Vulnerability: Potential Information Disclosure Through Error Messages
Approximate Line: 110, 138, 161, 189
Description: Error messages like `"Algo salio mal guardando el registro{categoria.Nombre}"` may leak sensitive information. The same holds for messages such as `"No se encontro la categoría con ID {categoriaId}"`.
Mitigation: Avoid including potentially sensitive information in error messages. Log detailed error information server-side, but return generic, user-friendly error messages to the client.
Improvement: Implement robust logging and monitoring to track errors without exposing internal details to end-users.
Code Quality Metrics
Complexity: Moderate
Description: The methods have a reasonable level of complexity, involving conditional statements, database interactions, and mapping operations. The controller itself is also moderately complex as it handles many routes.
Improvement: Consider using more advanced techniques like MediatR to decouple controller actions from

business logic. Introduce additional abstraction layers in the repository for complex queries.
Duplication: High
Description: There is significant duplication in the `ActualizarPatchCategoria` and `ActualizarPutCategoria` methods (model validation, existence check, mapping, error handling). The error handling is also generally repetitive throughout the controller.
Improvement: Refactor the common logic into reusable helper methods or filters. Use custom model binders to handle complex model validation logic. Employ exception filters to centralize error handling.
Readability: Good
Description: The code is generally readable, with clear variable names and comments. However, excessive comments may indicate that the code could be more self-documenting.
Improvement: Reduce redundancy by improving the logic and method extraction, rather than relying on comments, or reduce the density of the comments for a less verbose output.
Coupling: High
Description: The controller is tightly coupled to the `ICategoriaRepositorio` and `IMapper` interfaces. While dependency injection is used, the controller still directly interacts with these dependencies, potentially making it difficult to test and maintain.
Improvement: Introduce an additional layer of abstraction, such as a service layer, between the controller and the repository. This would decouple the controller from the data access logic and improve testability.
Proposed Solution

The proposed solution involves several steps:

1. **Implement authorization:** Re-enable `[Authorize]` attributes on appropriate endpoints with relevant roles or policies.
2. **Address IDOR vulnerability:** In the repository methods, verify that the user has access to modify the specified category based on their context.
3. **Mitigate Mass Assignment:** Introduce separate DTOs for creating and updating categories, exposing only the necessary properties.
4. **Refactor Duplicated Code:** Create helper methods to encapsulate common validation and error handling logic.
5. **Centralized Error Handling:** Use exception filters to handle exceptions globally and return standardized error responses.
6. **Reduce Information Disclosure:** Ensure detailed error information is logged server-side, but only generic messages are returned to the client.
7. **Introduce Service Layer:** Decouple the controller from the repository by introducing a service layer that handles business logic.

```
```