Reporte de Vulnerabilidad

Archivo: IPeliculaRepositorio.cs

Código Analizado:

```
?using ApiPeliculas.Modelos;

namespace ApiPeliculas.Repositorio.IRepositorio
{
    public interface IPeliculaRepositorio
    {
        //V1
        //ICollection<Pelicula> GetPeliculas();

        //V2
        ICollection<Pelicula> GetPeliculas(int pageNumber, int pageSize);
        int GetTotalPeliculas();
        ICollection<Pelicula> GetPeliculasEnCategoria(int catId);
        IEnumerable<Pelicula> BuscarPelicula(string nombre);
        Pelicula GetPelicula(int peliculaId);
        bool ExistePelicula(int id);
        bool ExistePelicula(string nombre);

        bool CrearPelicula(Pelicula pelicula);
        bool ActualizarPelicula(Pelicula pelicula);
        bool BorrarPelicula(Pelicula pelicula);
        bool Guardar();
    }
}
```

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

El código proporcionado define una interfaz `IPeliculaRepositorio` para interactuar con datos de películas. A continuación, se presenta un análisis de seguridad y calidad del código.

Vulnerabilidades de Seguridad

Si bien el código proporcionado es una interfaz y no contiene la implementación, es importante considerar las posibles vulnerabilidades que podrían surgir en la implementación de esta interfaz, especialmente en el repositorio concreto que la implemente.

- **Tipo: Inyección SQL (Potencial)**
  Línea Aproximada: La vulnerabilidad no está directamente en este código, pero es un riesgo al implementar la función `BuscarPelicula(string nombre)` en el repositorio concreto.
  Descripción: Si el parámetro `nombre` en la función `BuscarPelicula` se usa directamente en una consulta SQL sin la debida sanitización o parametrización, podría ser vulnerable a inyección SQL. Un atacante podría inyectar código SQL malicioso a través de este parámetro.
  Mitigación:
    - Utilizar siempre consultas parametrizadas o ORM (Object-Relational Mapper) como Entity Framework Core que manejan la parametrización automáticamente.
    - Validar y sanear la entrada `nombre` antes de utilizarla en la consulta. Considerar una lista blanca de caracteres permitidos.
- **Tipo: Ataque de Denegación de Servicio (DoS) (Potencial)**
  Línea Aproximada: Relacionado con `GetPeliculas(int pageNumber, int pageSize)` y `BuscarPelicula(string nombre)`.
  Descripción: Si `pageSize` en `GetPeliculas` o la búsqueda en `BuscarPelicula` no están controlados adecuadamente, un atacante podría solicitar un número extremadamente alto de resultados, causando un alto consumo de recursos en el servidor y potencialmente un ataque DoS.
  Mitigación:

    - Limitar el valor máximo de `pageSize` a un valor razonable.
    - Implementar paginación adecuada y limitar el número de resultados por página.
    - Monitorizar el uso de recursos del servidor para detectar anomalías.
- **Tipo: Exposición de Información Sensible (Potencial)**
  Línea Aproximada: Relacionado con la implementación de `GetPelicula(int peliculaId)`.
  Descripción: Si la implementación de `GetPelicula` devuelve información sensible sobre la película (por ejemplo, información de derechos de autor, datos de reparto con información personal, etc.) sin la autorización adecuada, podría considerarse una exposición de información.
  Mitigación:
    - Implementar mecanismos de control de acceso para restringir el acceso a información sensible.
    - Considerar la posibilidad de enmascarar o anonimizar datos sensibles si no son necesarios para todos los usuarios.

Métricas de Calidad del Código

El código actual es una interfaz, por lo que algunas métricas no son directamente aplicables. Sin embargo, podemos considerar aspectos relacionados con su diseño:

- **Complejidad:** La interfaz es relativamente simple y enfocada, lo cual es bueno. La complejidad se trasladará a la implementación del repositorio.
- **Acoplamiento:** La interfaz `IPeliculaRepositorio` está acoplada al modelo `Pelicula`. Esto es inevitable, pero es importante mantener la interfaz lo más genérica posible para evitar acoplamientos innecesarios.
- **Legibilidad:** La interfaz es muy legible y fácil de entender. Los nombres de los métodos son descriptivos y el uso de tipos de datos estándar contribuye a la legibilidad.
- **Duplicación:** No hay duplicación en la interfaz en sí. Sin embargo, la implementación del repositorio podría incurrir en duplicación si no se abstraen las operaciones comunes.
- **Mantenibilidad:** La interfaz es fácil de mantener debido a su simplicidad y claridad. Cualquier cambio en la implementación del repositorio no afectará la interfaz a menos que se requieran nuevas funcionalidades.

Solución Propuesta

Las siguientes son algunas propuestas para mejorar la seguridad y la calidad del código, centradas en cómo debería ser implementada la interfaz.

- **Implementación Segura de `BuscarPelicula`:**
    - Utilizar Entity Framework Core (u otro ORM) con consultas parametrizadas para evitar inyección SQL.
    - Validar y sanear la entrada `nombre`.
- **Control de Paginación:**
    - Definir un valor máximo para `pageSize` en la implementación de `GetPeliculas`.
    - Implementar validación de entrada para `pageNumber` y `pageSize` para prevenir valores negativos o excesivamente grandes.
- **Control de Acceso:**
    - Implementar roles y permisos para controlar el acceso a la información de las películas.
    - Evitar exponer información sensible innecesariamente.
- **Abstracción en la Implementación:**
    - Crear métodos privados para abstraer operaciones comunes en el repositorio para evitar duplicación de código.
- **Manejo de Errores:**
    - Implementar un manejo de errores robusto para capturar excepciones y devolver mensajes de error informativos.
    - Utilizar logging para registrar errores y facilitar la depuración.

```
```